UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

Entwicklung einer mobilen Anwendung zur Registrierung von NFC-Tags für die Identifizierung von Personen

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.) im Studiengang
Medizinische Informatik

vorgelegt von

Julian Suleder

27. Juli 2015

Referent: Prof. Dr.-Ing. Daniel Pfeifer

Korreferent: Prof. Dr. Martin Haag

Betreuer: Dipl.-Inform. Med. Peter Rohn

Zusammenfassung

Die Identifizierung von Angehörigen der Hochschule Heilbronn erfolgt in der Regel über die Kombination aus Benutzername und Passwort. In verschiedenen Einsatzgebieten, wie z.B. an einer Parkschranke, ist eine Eingabe der Benutzermerkmale nicht möglich oder hinderlich. Hierfür soll die Mensakarte des Studentenwerks Heidelberg als identifizierendes Merkmal erschlossen werden. Dies macht die Verknüpfung von Benutzerkonto und Karte notwendig.

Im Rahmen dieser Bachelorarbeit werden zunächst verschiedene Umsetzungsmöglichkeiten für mobile Anwendungen zur Verknüpfung von Benutzer und Karte unter Verwendung der NFC-Technologie analysiert und ausgewertet. Anschließend wird ein funktionaler Prototyp für Smartphones der *Android*-Plattform entwickelt, der die einfache Einbindung weiterer Funktionalität ermöglichen soll.

Der entwickelte Prototyp ist im Hochschulnetz testweise für die Registrierung und Deregistrierung von NFC-Karten einsetzbar. Vor einer realen Nutzung des Systems müssen der Datenschutz und andere organisatorische und rechtliche Pflichten, wie zum Beispiel das Telemediengesetz, berücksichtigt werden.

Danksagung

An dieser Stelle möchte ich mich bei meinem Referenten Herrn Prof. Dr. Daniel Pfeifer für die engagierte Betreuung und die vielen hilfreichen Gespräche bedanken.

Ein weiterer Dank geht an Herrn Peter Rohn für das Feedback und die Motivation während der Bearbeitung des Themas.

Für die Bereitstellung von Server-Infrastruktur und dabei geleistete technische Unterstützung möchte ich den Administratoren der Medizinischen Informatik und speziell Herrn Daniel Zsebedits danken.

Dank gilt weiterhin allen Personen, die mich bei dieser Arbeit unterstützt, oder während des Studiums begleitet haben.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CDI	Contexts and Dependency Injection
CORS	Cross-Origin Resource Sharing
CSP	Content Security Policy
CSS	Cascading Style Sheets
DTO	Datentransferobjekt
HATEOAS	Hypermedia As The Engine Of Application State
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JAXB	Java Architecture for XML Binding
JAX-RS	Java API for RESTful Web Services
JPA	Java Persistence API
JSON	JavaScript Object Notation
MBaaS	Mobile Backend as a Service
NFC	Near Field Communication
ORM	Object/Relational-Mapper
REST	Representional State Transfer
RFID	Radio-Frequency Identification
RPC	Remote Procedure Call
SDK	Software Development Kit
SoC	Separation of Concerns
SOP	Same-Origin Policy
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSRF	Cross-Site-Request-Forgery
XSS	Cross-Site Scripting

1 Einleitung

1.1 Motivation

Im Zuge des Ausbaus des Bildungscampus am Europaplatz in Heilbronn bestehen eingeschränkte Parkmöglichkeiten für Studierende und Mitarbeiter. In naher Zukunft werden neue Parkplätze geschaffen, die exklusiv für die genannten Personen nutzbar sein sollen, wofür diese identifizierbar sein müssen.

Die Identifizierung der Personen soll über das Benutzerkonto der Hochschule Heilbronn vorgenommen werden. Hierbei kann die Mensakarte des Studentenwerks Heidelberg als identifizierendes Merkmal verwendet werden. Diese ist noch nicht mit den Benutzerkonten verknüpft, was nun durchgeführt werden soll. Die Verknüpfung soll von jedem Benutzer selbst durchführbar sein und keine zusätzlichen Anschaffungen, wie z.B. Registrierungsterminals erforderlich machen.

Da die Mensakarte die Near Field Communication (NFC)-Technologie unterstützt, welche seit geraumer Zeit, z.B. für das bargeldlose Bezahlen, genutzt wird, und viele Smartphones bereits einen NFC-Sensor besitzen, soll in dieser Bachelorarbeit eine mobile Applikation zur Registrierung dieser NFC-Karten entwickelt werden.

1.2 Ziele

Die Ziele dieser Bachelorarbeit sind in der nachfolgenden Aufzählung erläutert:

1. Das Erstellen eines Konzepts für die Umsetzung einer mobilen Anwendung zur Registrierung von NFC-Tags und Benutzerkonten. Die zu entwickelnde mobile Anwendung soll als hybride App konzipiert sein und für die Android-Plattform entwickelt werden.
2. Die Analyse der verfügbaren Umsetzungsmöglichkeiten hybrider Applikationen.
3. Die Analyse der Umsetzungsmöglichkeiten von REST-Services mit Java-Standards (JAX-RS 2.0).
4. Die Implementierung eines funktionalen Prototyps des nach 1. entwickelten Konzepts mit den nach 2. und 3. ausgewählten Technologien.

1.3 Vorgehensweise und Struktur der Arbeit

Zu Beginn der Arbeit wurde eine Anforderungsanalyse durchgeführt (Kapitel 3). Die ermittelten Anforderungen dienten als Grundlage für die Analyse und Evaluation verfügbarer Technologien (Kapitel 4), mit welchen anschließend ein Entwurf (Kapitel 5) erstellt wurde. Dieser wurde in der Folge (Kapitel 6) umgesetzt.

2 Grundlagen

Dieser Abschnitt stellt die für die Entwicklung einer hybriden Applikation notwendigen Konzepte und Frameworks zur Realisierung von hybriden Applikationen und REST-Services vor.

2.1 NFC - Near Field Communication

Die NFC-Technologie ermöglicht eine einfache, standardisierte [1] und sichere Zwei-Wege-Interaktion zwischen elektronischen Geräten, sogenannte *Tags*, wodurch die Verbraucher kontaktlose Transaktionen durchführen und Zugriff zu digitalen Inhalten erhalten können. NFC ergänzt durch die Nutzung bestehender Standards für kontaktlose Kartentechnologien[1] viele verbrauchernahe Funktechnologien, wie z.B. die Radio-Frequency Identification (RFID). Es verträgt sich mit der bestehenden Infrastruktur für kontaktlose Karten, was die Nutzung eines Geräts in verschiedenen Systemen ermöglicht. Die Übertragungsreichweite beträgt maximal 4 Zentimeter [2].

2.2 REST - Representional State Transfer

Roy Fielding veröffentlichte in seiner Doktorarbeit[2] eine Sammlung von Architekturprinzipien für die Entwicklung verteilter Anwendungen. Representional State Transfer (REST) ist eine Teilmenge dieser Prinzipien. In der Praxis werden die Prinzipien meist als Beschränkungen an Services mit dem Hypertext Transfer Protocol (HTTP) umgesetzt. REST definiert folgende Bedingungen:

1. **Adressierbarkeit:** Ein Datensatz wird zu einer Ressource abstrahiert, die über ihren eindeutigen Uniform Resource Identifier (URI) adressierbar ist [3].
2. **Unterschiedliche Repräsentationen:** Eine Ressource kann in verschiedenen Formaten repräsentiert werden. Je nach Programm wird eine andere Repräsentation, z.B. in HyperText Markup Language (HTML) oder JavaScript Object Notation (JSON), erwartet [3].
3. **Zustandslosigkeit:** Die Kommunikation zwischen Client und Server ist zustandslos. Das bedeutet, dass Anfragen alle für ihre Behandlung nötigen Informationen enthalten müssen [3].
4. **HATEOAS:** Mit Hypermedia As The Engine Of Application State (HATEOAS) werden Ressourcen in Sammlungen durch deren URI ersetzt und vom Client einzeln nachgeladen. Zusätzlich werden weitere mögliche Aktionen in Form von Verweisen mitgeteilt, was zur Vereinfachung von der Client-Logik führt [3].

[1] ISO/IEC 14443-Familie: Internationale Normenreihe für kontaktlose Chipkarten.

[2] Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures: `https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`, 2000.

2.3 JAX-RS - Java API for RESTful Web Services

Java API for RESTful Web Services (JAX-RS) ist ein Application Programming Interface (API), welches die Konzepte von REST in Java vereinheitlicht und als Standard einen Rahmen für das Entwickeln von RESTful Web Services bildet. Der Standard sieht vor, das Verhalten von Services durch Annotationen zu spezifizieren.

Eine Menge von Services benötigt immer eine `Application`, die den Basispfad der Anwendung festlegt. Der Zugriff auf die in Listing 2.1 definierte Applikation erfolgt über `http://<server>/api/rest/`.

```
1 import javax.ws.rs.ApplicationPath;
2 import javax.ws.rs.core.Application;
3
4 @ApplicationPath("/api/rest")
5 public class RestApplication extends Application {
6   public RestApplication() {}
7 }
```

Listing 2.1: Ein RESTful Web Service mit JAX-RS spezifiziert durch eine Applikation, den Basispfad für den Zugriff auf Services.

In Listing 2.2 ist die Methode eines Service definiert. Sie ist durch die HTTP-Operation über den angegebenen Pfad aufrufbar. Zusätzlich werden die Formate, die sie konsumiert und produziert und die aus der Anfrage genutzten Parameter, optional mit Standardwerten, angegeben.

```
1 import javax.ws.rs.*;
2
3 @Path("1.0/persons")
4 public class PersonService {
5     @GET
6     @Path("/person/{id}")
7     @Produces(MediaType.APPLICATION_JSON)
8     public Person getPerson(@PathParam("id") int id, @DefaultValue
          ("Max Mustermann")@QueryParam("name") String name)
9         Person person = new Person();
10        person.setName(name);
11        return person;
12    }
13 }
14 public class Person{
15    private String name = "";
16    public String getName(){ return name; };
17    public void setName(String name){ this.name=name; }
18 }
```

Listing 2.2: Eine Service-Methode die über GET-Anfragen aufgerufen wird und JSON produziert. Die Übersetzung von Objekt zu JSON wird mit Providern (Abschnitt 4.2.2) vorgenommen.

2.4 Hybride Applikationen

Eine hybride Applikation, auch *hybride App* genannt, ist eine mit gängigen Webtechnologien geschriebene Web-Applikation für mobile Geräte, die lokal in einem Browserfenster, einer sogenannten *Web View*, gestartet wird. Diese besitzt keine visuelle Navigation und Adresseingabe, kann jedoch durch Erweiterungen auf die Gerätefunktionalität zugreifen. Das bekannteste Framework für hybride Apps ist Phonegap, an dessen Beispiel in Abschnitt 2.6.2 der Aufbau einer hybriden App erklärt wird.

2.5 Sicherheitsprobleme hybrider Applikationen

Auf Web-Anwendungen abzielende Angriffe bedrohen hybride Apps in gleichem Maße. Im Folgenden werden die zwei größten Bedrohungen und clientseitigen Gegenmaßnahmen zur Erschwerung eines Angriffes vorgestellt.

- **Problem:** Bei einer Cross-Site-Request-Forgery (XSRF) wird das Vertrauen der Webanwendung in den Benutzer ausgenutzt. Da HTTP zustandslos ist, muss der Benutzer bei jeder Aktion auch Daten zur Authentifizierung mitsenden. Dies machen sich Angreifer bei XSRF-Angriffen zu Nutze. Sie manipulieren, z.B. durch einen Man-in-the-Middle-Angrif, den Code der Webseite. Der Benutzer führt anschließend unwissentlich schädliche Aktionen, z.B. eine Bestellung, aus.
- **Lösung:** Damit keine Skripte oder manipulierte HTML-Seiten bezogen werden können, wird mittels Same-Origin Policy (SOP) die Quelle von Ressourcen strikt auf den Pfad der Anwendung beschränkt. Um darüber hinaus Zugriff auf weitere Ressourcen zu gestatten, werden vertrauenswürdige Quellen durch Cross-Origin Resource Sharing (CORS) in einer Whitelist erfasst. Der Client überprüft damit bereits vor der Anforderung der Ressource deren Vertrauenswürdigkeit [4].
- **Problem:** Eine Web-Anwendung ist für Cross-Site Scripting (XSS) anfällig, wenn sie Daten als Text von einem Server bezieht. Zur Ausnutzung wird in den Daten ein `script`-Element gesendet. Dazu muss der Angreifer die Antwort abfangen und manipulieren, oder den Server übernehmen. Bei Ausführung auf dem Client, kann der Angreifer sensible Daten des Benutzers stehlen und ausnutzen [5, S. 133ff].
- **Lösung:** Um dies zu verhindern, werden Eingaben maskiert, d.h. gefährliche Funktionszeichen durch ungefährliche Äquivalente ersetzt (z.B. `<` durch `<`). Diese Ersetzung wird erst bei der Darstellung durch die Web View rückgängig gemacht und verhindert die Ausführung von Code während der Verarbeitung. Werden Skripte aktiv nachgeladen, so müssen die Quellen zusätzlich zur SOP ebenfalls in einer Content Security Policy (CSP) Whitelist erfasst werden. Anderenfalls wird die Ausführung des Codes verhindert.

2.6 Middleware-Frameworks

In diesem Abschnitt werden Frameworks für die Realisierung oder Erweiterung der Schnittstelle zwischen nativem Betriebssystem und hybrider Anwendung vorgestellt.

2.6.1 AngularJS

AngularJS ist eine Open-Source-JavaScript-Bibliothek, die von Google weiterentwickelt und unterhalten wird. Die für das Projekt wichtigsten Eigenschaften sind die leichte Erweiterbarkeit und Modularisierung von AngularJS-Applikationen, Dependency Injection, sowie die Zwei-Wege-Datenbindung. Die Zwei-Wege-Datenbindung ermöglicht eine automatische Aktualisierung von Variablen-Werten bei Änderungen an Oberfläche und Model. Es erweitert HTML um die Wiederverwendung von Code-Ausschnitten zu erhöhen. Dadurch wird die Grundlage geschaffen, komplexe Anwendungen komfortabler und leichter zu erstellen [6, S. 3].

Eine AngularJS-Applikation gliedert sich in Module. Jedes Modul ist eine eigenständige Applikation und in AngularJS mit eindeutigem Namen, Navigation, Konstanten und Controllern definiert. Ein Modul kann Abhängigkeiten zu anderen Modulen besitzen, wofür es einen eindeutigen Namen tragen muss.

Die Navigation definiert die Zustände des Moduls. Zustände sind Webseiten und besitzen einen Namen und eine eindeutige URI zum Zugriff aus der Applikation. Zusätzlich wird eine HMTL-Seite für die Darstellung des Zustands und ein Controller für die Datenhaltung und Logik angegeben.

Die Ressourcen der Applikation müssen vom Browser beim Start geladen werden können. Hierzu definiert jede AngularJS-Applikation eine `index.html`-Seite. In ihr werden im *head* alle JavaScript- und Cascading Style Sheets (CSS)-Files der Applikation angegeben und im *body* eine AngularJS-Applikation mit einer Einstiegsseite definiert (siehe Listing 2.3).

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <script src="angular.js"></script>
5         <script src="app.js"></script>
6     </head>
7     <body ng-app="app">
8         <!--some nice web-page-->
9     </body>
10 </html>
```

Listing 2.3: Beispiel der `index.html` einer AngularJS-Applikation.

2.6.2 Apache Cordova/ Phonegap

Apache Cordova/ Phonegap ist eine Zusammenstellung von Geräte-APIs, die einem Entwickler mobiler Apps Zugriff zu nativen Gerätefunktionen, wie z.B. der Kamera oder dem Beschleunigungsmesser, mit JavaScript ermöglichen. In Kombination mit einem UI-Framework erlaubt es eine Smartphone-App mit HTML, CSS und JavaScript zu entwickeln [7]. Im Oktober 2012 wurde PhoneGap in die Apache Software Foundation (ASF) unter dem Namen Apache Cordova integriert und erhielt Top-Level-Projekt-Status [8].

Die Architektur einer Cordova-Applikation ist in Abbildung 2.1 dargestellt. Die hybride App besteht aus generischen Anteilen der Cordova-API und anwendungspezifischem Code. Die Anwendung wird in der Web View geladen und angezeigt. Über das `cordova.js`-File erhält sie Zugriff auf grundlegende native Gerätefunktionalitäten und erweiternde Cordova-Plugins. Diese beinhalten einen nativen und einen JavaScript-Anteil. Nativer plattformspezifischer Code wird über den generischen JavaScript-Code abstrahiert, wodurch die Komplexität der nativen Implementierungen verborgen wird. Dies hat zur Folge, dass die Applikation rein mit gängigen Webstandards programmiert werden kann. Beim Build wird ein Artefakt für jede gewünschte Plattform erzeugt. Die fertige App wirkt dabei, bis auf die HTML5-Oberfläche, nahezu vollständig nativ [9, S. 191f]. Für die Entwicklung wird das native Software Development Kit (SDK) einer Plattform benötigt (bspw. AndroidStudio für Android- und Xcode für iOS-Entwicklung).

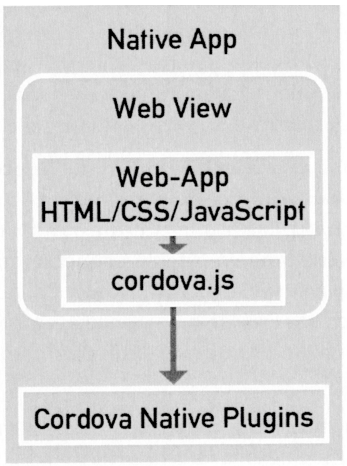

Abbildung 2.1: Architektur einer Cordova-Applikation [9, S. 192].

2.6.3 Appcelerator

Appcelerator ist ein kostenpflichtiges Framework zur Entwicklung hybrider Applikationen, das Services wie z.B die Bereitstellung kompletter Cloud-Backends für mobile Applikationen (Mobile Backend as a Service (MBaaS)) oder App-Analyse-Methoden anbietet [10]. Auf Grund der Kosten und mangelnder NFC-Schnittstelle wurde das Framework ausgeschlossen.

2.6.4 FireMonkey

FireMonkey ist ein kostenpflichtiges Framework für die Entwicklung geräteübergreifender, nicht nur mobiler, nativer Anwendungen unter Verwendung von HTML5, Delphi oder C++ [11]. Auf Grund der Kosten wurde das Framework ausgeschlossen.

2.6.5 Intel XDK

Intel XDK ist eine kostenfreie Entwicklungsumgebung von Intel für die Entwicklung von Cordova-Apps. Die Entwicklungsumgebung enthält einige hilfreiche Werkzeuge und Erweiterungen für z.B. die Entwicklung von Spielen oder Einbindung von Multimedia-Inhalten [12].

2.6.6 NativeScript

NativeScript ist ein Framework für mobile Applikationen, das es Entwicklern ermöglicht, native Apps für iOS und Android zu erstellen, ohne plattformspezifischen Code schreiben zu müssen [13]. Auf Grund mangelnden Zugangs zum NFC-Sensor wurde das Framework ausgeschlossen.

2.6.7 Sencha Touch

Sencha Touch ist ein Framework zur Entwicklung hybrider Applikationen, dass über die Cordova-Funktionalität hinaus weitere Gerätefunktionalität bereitstellt. Es vereinfacht die Realisierung performanter, nativ aussehender Applikationen über vorgefertigte Erweiterungen [14]. Auf Grund der Kosten wurde das Framework ausgeschlossen.

2.6.8 Trigger.IO

Trigger.IO ist ein Framework für mobile Applikationen. Es abstrahiert ähnlich wie Cordova plattformspezifische Implementierungen in Bibliotheken, die über JavaScript generisch nutzbar sind [15]. Auf Grund der Kosten und mangelnden Zugang zum NFC-Sensor wurde das Framework ausgeschlossen.

2.6.9 Xamarin

Xamarin ist ein Framework für die Entwicklung mobiler Applikationen mit C#. Es bietet Services zum automatischen Build, Testen und App-Monitoring an. Zusätzlich verfügt es über umfassende Schnittstellen um den Backend-Zugriff mit verschiedensten Technologien zu vereinfachen [16]. Auf Grund der Kosten wurde das Framework ausgeschlossen.

2.7 UI-Frameworks

In diesem Abschnitt werden Frameworks für hybride Applikationen vorgestellt, die sich auf die Oberfläche spezialisiert haben. Diese werden für die Entwicklung in Verbindung mit einem Middleware-Framework (Abschnitt 2.6) verwendet oder beinhalten dieses.

2.7.1 AppGyver

AppGyver ist ein kostenpflichtiges, auf dem Ionic Framework (2.7.3) basierendes Framework für hybride Applikationen, das besonderen Wert auf die Nativität der Benutzeroberfläche legt [17]. Auf Grund der Kosten wurde das Framework ausgeschlossen.

2.7.2 Famo.us

Famo.us ist ein Framework für die Entwicklung von Webseiten mit spieleähnlichen Animationen und Visualisierungen mit Webstandards [18]. Das Framework wurde ausgeschlossen, da es zwar mit Cordova für hybride Applikationen verwendet werden kann, aber primär für Webseiten gedacht ist.

2.7.3 Ionic Framework

Ionic ist ein umfangreiches HTML5 Framework für die Entwicklung von hybriden Applikationen. Es fokussiert sich auf die Benutzeroberfläche der Applikation und baut auf Phonegap und AngularJS auf. Das quelloffene Framework beinhaltet zahlreiche Tools zur Unterstützung der effizienten Entwicklung und Personalisierung der Applikation, wie zum Beispiel die Automatisierung des Build-Prozesses [19]. Die zu entwickelnde hybride Applikation wurde auf Grund dieser Vorteile mit Ionic umgesetzt.

2.7.4 jQuery Mobile

jQuery Mobile ist ein User Interface (UI)-Framework für mobile Webanwendungen. In Verbindung mit Phonegap/Cordova können hybride Applikationen mit komplexen Oberflächen erstellt werden [20].

2.7.5 Kendo UI

Kendo UI ist ein auf jQuery basierendes Framework für die Entwicklung hybrider Applikationen mit optionaler Einbindung von AngularJS oder Twitter Bootstrap[3]. Es bringt viele vorgefertigte UI-Bausteine und Werkzeuge mit [21]. Auf Grund der Kosten wurde das Framework ausgeschlossen.

2.7.6 Onsen UI

Onsen UI ist ein Open-Source HTML5 Oberflächen-Framework, das einerseits für Websites, andererseits unter Verwendung von AngularJS und Phonegap/Cordova für hybride Applikationen verwendet werden kann. Es soll nach eigenen Angaben performanter als andere UI Frameworks sein [22].

[3]Twitter Bootstrap: `http://getbootstrap.com`

2.8 REST-Frameworks

In diesem Abschnitt werden die verbreitetsten Implementierungen des JAX-RS-Standards vorgestellt.

2.8.1 Apache CXF

Die REST-Erweiterung für das umfangreiche Webservice-Framework Apache CXF unterstützt viele Transportprotokolle und zusätzliche Module für die Entwicklung von REST-Services. Das Framework ist darauf ausgelegt, möglichst viele Datenformate, Transportprotokolle und Standards zu unterstützen [23].

2.8.2 Jersey

Die JAX-RS-Implementierung Jersey ist die Referenzimplementierung des Standards und bietet zusätzliche Werkzeuge für die Entwicklung von REST-Services und -Clients. Das Framework wird mit den Updates des Standards weiterentwickelt [24].

2.8.3 RESTEasy

RESTEasy ist ein Framework von JBoss für die Entwicklung von RESTful Web Services. Es ist eine vollständig konforme Implementierung des JAX-RS Standards. Es bietet über diesen Standard hinaus weitere umfangreiche Funktionen, wie z.B. eine Client-API und viele Provider für Transportprotokolle und Datenformate [25]. Es wurde für die Implementierung des REST-Backends der hybriden Applikation ausgewählt.

2.8.4 Restlet Framework

Das Restlet Framework war das erste Framework zur Entwicklung von REST-Services, bevor diese populär wurden und kann in einer Vielzahl von Umgebungen verwendet werden. Es integriert den JAX-RS Standard als Erweiterung des Frameworks und ermöglicht es, Anwendungen nicht zwangsläufig auf einem Anwendungsserver einsetzen zu müssen [26].

3 Anforderungsanalyse

In diesem Kapitel wird zunächst der grundlegende Gedanke der Applikation (Abschnitt 3.1) erläutert und deren Nutzungskontext (Abschnitt 3.2) beschrieben. Dem folgen die Anforderungen an das System, welche mit den Papierentwürfen der Benutzerschnittstelle im Abschnitt 3.5 die Grundlage für die Technologieevaluation in Kapitel 4 bilden.

3.1 Kerngedanken der Applikation

Nach den in Abschnitt 1.2 vorgestellten Zielen soll eine hybride Applikation für die Android-Plattform entwickelt werden. Der Nutzer liest mit der Applikation über den NFC-Sensor seines Smartphones die Seriennummer seiner Mensakarte oder einem beliebigen anderen NFC-Tag aus und verknüpft diese mit seinem Benutzerkonto. Diese Verknüpfung kann er, z.B. bei Verlust der Karte, rückgängig machen (vgl. Abbildung 3.1).

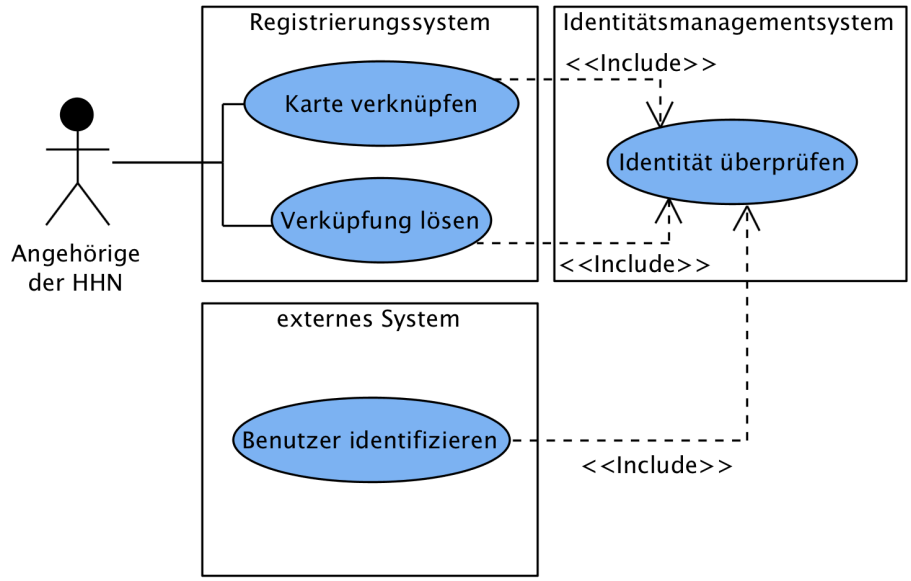

Abbildung 3.1: Die Verknüpfung einer Karte und die Lösung dieser Verknüpfung sind die primären Anwendungsfälle an das System. Um diese Aktionen durchführen zu können, muss sich der Benutzer als Angehöriger der HHN ausweisen können. Externe Systeme können die Benutzer über die Karten identifizieren.

3.2 Nutzungskontext

Der Nutzungskontext beinhaltet benutzer- und systemspezifische Merkmale:

- **Benutzermerkmale:** Der Benutzer ist in der Hochschule Heilbronn als Mitarbeiter oder Student registriert und besitzt ein aktives Benutzerkonto sowie eine Mensakarte des Studentenwerks Heidelberg. Da die Benutzergruppe hinsichtlich ihrer technischen Affinität heterogen zusammengesetzt ist, wird der Benutzer im Allgemeinen als Laie angesehen.
- **Umgebung des Systems:** Der Zugriff auf das System erfolgt via Internet über meist ungesicherte Zugänge, wie z.B. Mobile Netze oder öffentliche WLAN-Netzwerke. Die App soll vorwiegend für Smartphones entwickelt werden, die Bildschirmgröße ist nicht festgelegt.

3.3 Funktionale Anforderungen

Aus dem Nutzungskontext (Abschnitt 3.2) und den grundlegenden Gedanken hinter der App (Abschnitt 3.1) leiten sich folgende funktionale Anforderungen ab:

- Funktionaler Schwerpunkt der Applikation soll die Registrierung und Deregistrierung von NFC-Tags sein. Die Integration von Standardinhalten einer Applikation, wie zum Beispiel ein Impressum kann berücksichtigt werden, die Inhalte sind zu vernachlässigen.
- Der Benutzer soll testen können, ob sein Smartphone einen NFC-Sensor besitzt.
- Ist ein solcher Sensor vorhanden, soll der Benutzer durch das Auslösen einer Aktion nach einer Karte suchen. Dies soll so lange geschehen, bis eine Karte erkannt wurde, oder der Benutzer die Suche abbricht.
- Wird eine Karte erkannt, soll die Seriennummer der Karte automatisch ausgelesen werden.
- Der Benutzer soll die Seriennummer der Karte mit seinem Benutzerkonto der Hochschule verknüpfen können. Hierfür soll ein Formular mit Benutzernamen, Passwort und der Seriennummer der Karte vorhanden sein.
- Der Benutzer muss die Verknüpfung der Karte aktiv auslösen und Rückmeldung über Erfolg und Misserfolg erhalten. Ein Benutzer soll maximal eine Karte mit seinem Konto verknüpfen können. Ist eine Karte bereits verknüpft, soll diese durch die neue Karte ersetzt werden.
- Die Deregistrierung soll durch Eingabe von Benutzername und Passwort erfolgen.

3.4 Nichtfunktionale Anforderungen

- Der Zugriff auf die Karte ist in jedem Moment lesender Art.
- Die App muss auf Android-Smartphones und -Tablets bis August 2015 funktionieren und darf keine Kosten verursachen.
- Die Oberfläche soll vorerst nicht internationalisiert werden, aber leicht verständlich und gut bedienbar sein. Ein ansprechendes Design wird geringer priorisiert als die Funktionalität.
- Die Leistungsfähigkeit des Systems soll auch durch gleichzeitige Nutzung durch mehrere Benutzer in tolerierbarer Zeit erfolgen. Besondere Maßnahmen müssen dafür nicht umgesetzt werden.
- Da in naher Zukunft eine Integration weiterer Funktionalität in die App erfolgen soll, sollte der Entwurf der App eine Modularität und leichte Erweiterbarkeit aufweisen, dabei ist eine Versionierung von App und Schnittstelle zum Server Pflicht.
- Die Nutzung der App in unsicheren Netzwerken ist grundsätzlich risikobehaftet. Daher müssen Maßnahmen zur Absicherung hybrider Applikationen berücksichtigt werden.

3.5 Initiale Gestaltungslösung

In diesem Abschnitt werden als Ergebnis des Nutzungskontexts und der funktionalen Anforderungen aus Abschnitt 3.2 und 3.3 die initialen Gestaltungslösungen der Benutzeroberfläche für die Registrierung und Deregistrierung von NFC-Tags dargestellt.

3.5.1 Papierentwurf der Registrierung

In Abbildung 3.2 ist ein Zustandsdiagramm abgebildet, das die Zustände der Registrierung darstellt. Auf dessen Basis wurde ein Registrierungsassistent erstellt, der den Benutzer durch die Registrierung führt. Zunächst muss der Status des Sensors überprüft werden. Ist dieser vorhanden und aktiviert, kann die Suche nach einer Karte gestartet werden. Nach dem Auslesen werden die Benutzerdaten benötigt. Sind diese korrekt, so erfolgt die Verknüpfung.

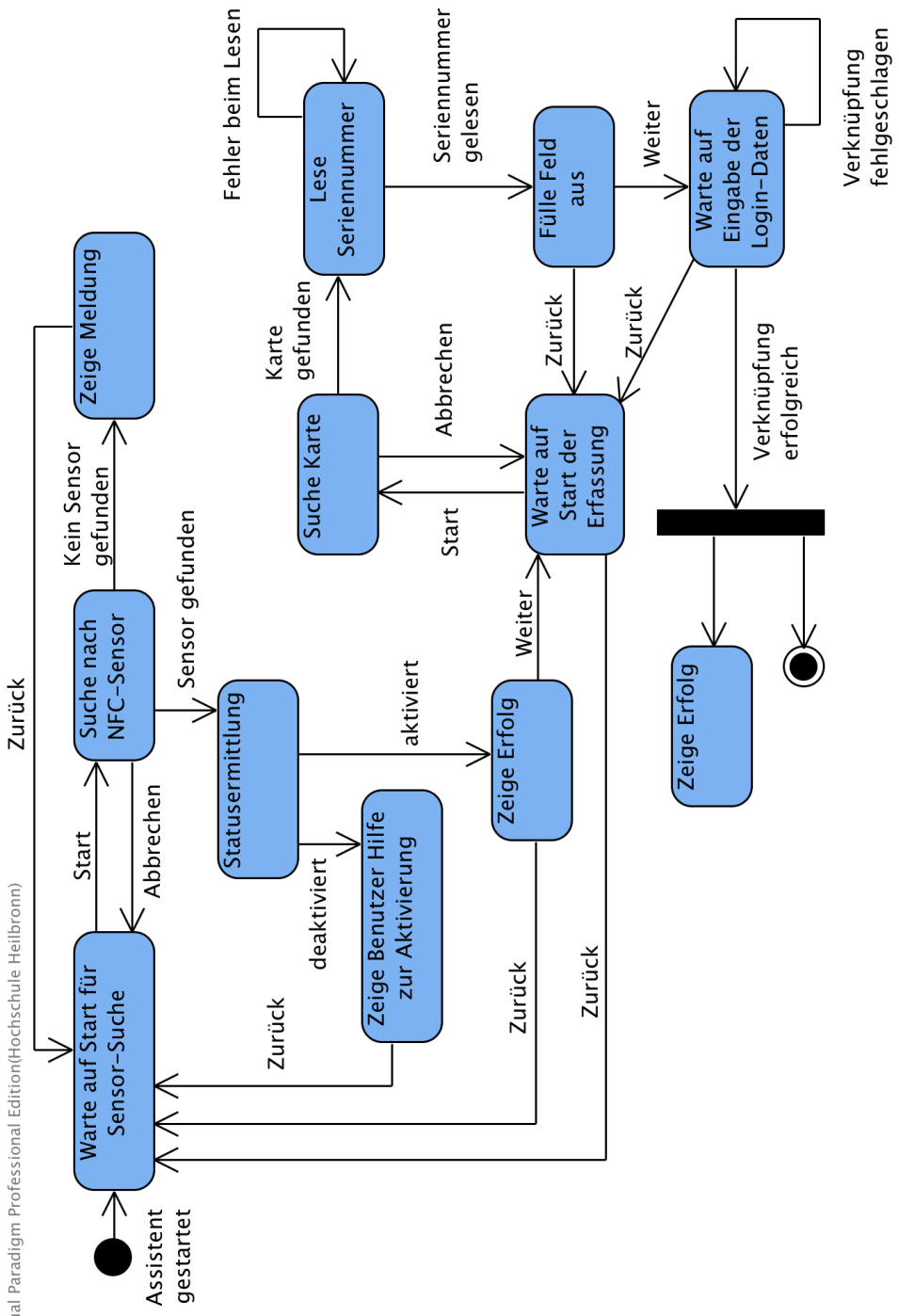

Abbildung 3.2: Das Zustandsdiagramm zeigt die Zustände der Registrierung von NFC-Tags mit den möglichen Alternativzuständen. Es wird angenommen, das sich der Status des Sensors während der Benutzung nicht ändert.

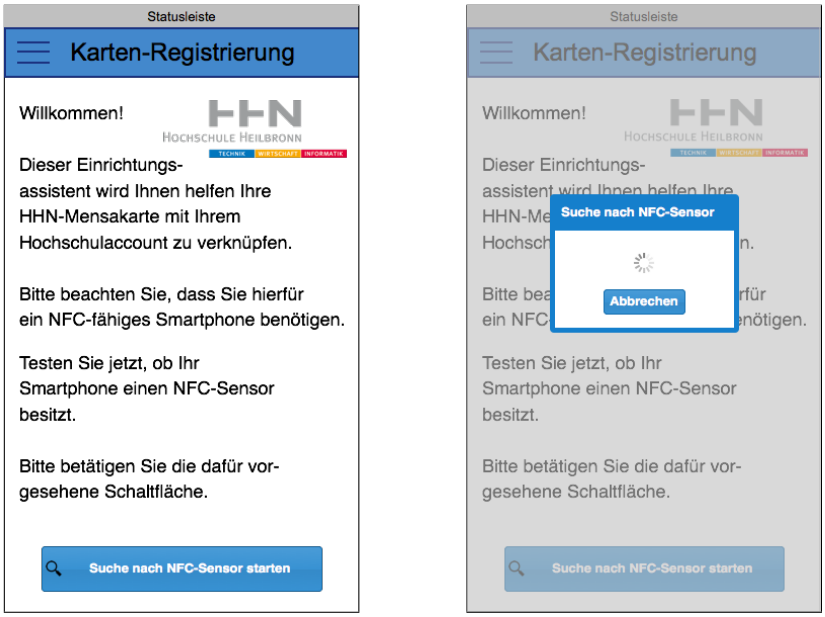

Abbildung 3.3: Papierbasierter Prototyp der Startseite des Registrierungsassistenten mit Hinweisen zum benötigten NFC-Sensor. In der rechten Abbildung ist die Statuserfassung des Sensors gestartet – ein modaler Dialog wird angezeigt. Dieser verschwindet, wenn der Status des Sensors erfasst ist.

Abbildung 3.4: Papierbasierter Prototyp der Meldungen, falls ein deaktivierter Sensor gefunden wurde (links), ein Fehler auftrat, oder kein Sensor gefunden wurde (rechts).

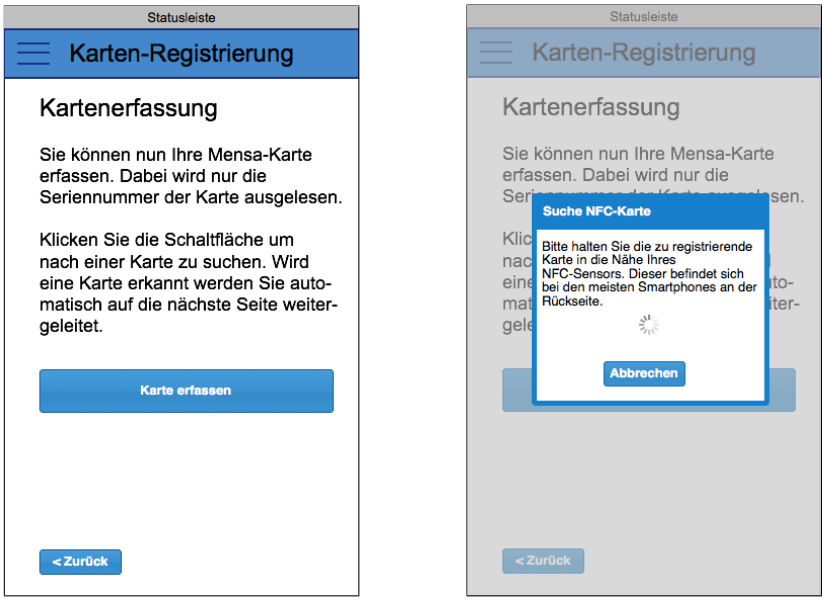

Abbildung 3.5: Prototyp der Kartenerfassung: Ein aktivierter Sensor muss gefunden worden sein. Beim Start der Erfassung wird so lange ein Dialog geöffnet, bis eine Karte gelesen oder die Erfassung abgebrochen wurde.

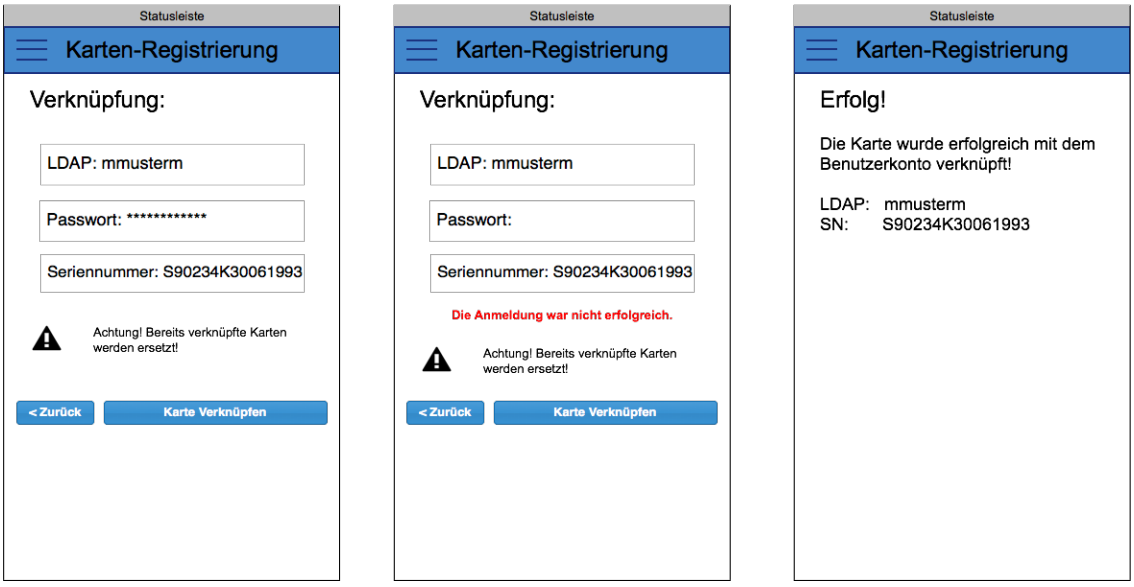

Abbildung 3.6: Prototyp des Absendeformulars der Registrierung: Der Nutzer muss Benutzername und Passwort eintragen. Die Seriennummer der eingelesenen Karte wird automatisch ausgefüllt. Sind die Benutzerdaten ungültig, so wird eine entsprechende Meldung angezeigt und das Passwortfeld geleert. War die Verknüpfung erfolgreich wird eine Meldung angezeigt.

3.5.2 Papierentwurf der Deregistrierung

In Abbildung 3.7 sind die möglichen Zustände bei der Lösung von Verknüpfungen zu sehen. Die daraus entwickelte Oberfläche besteht auf Grund der geringen Komplexität aus einem einzigen Formular.

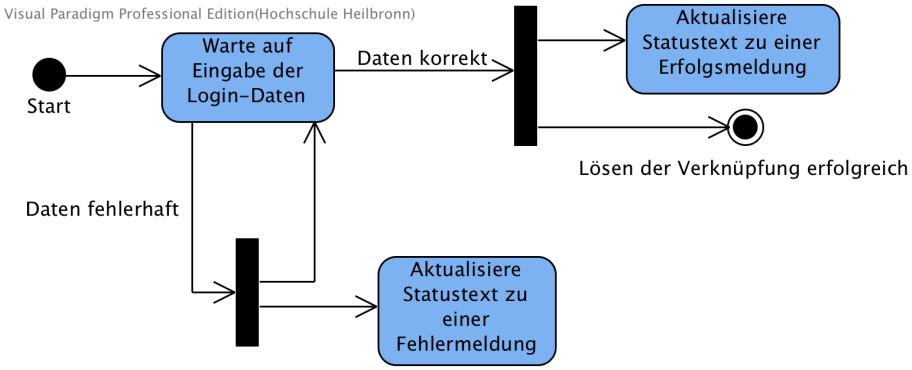

Abbildung 3.7: Zustände bei der Lösung der Verknüpfung: Nach dem Versenden der Benutzerdaten wird der Statustext aktualisiert.

In den nachfolgenden Papierentwürfen ist die Oberfläche für die Deregistrierung der NFC-Tags dargestellt.

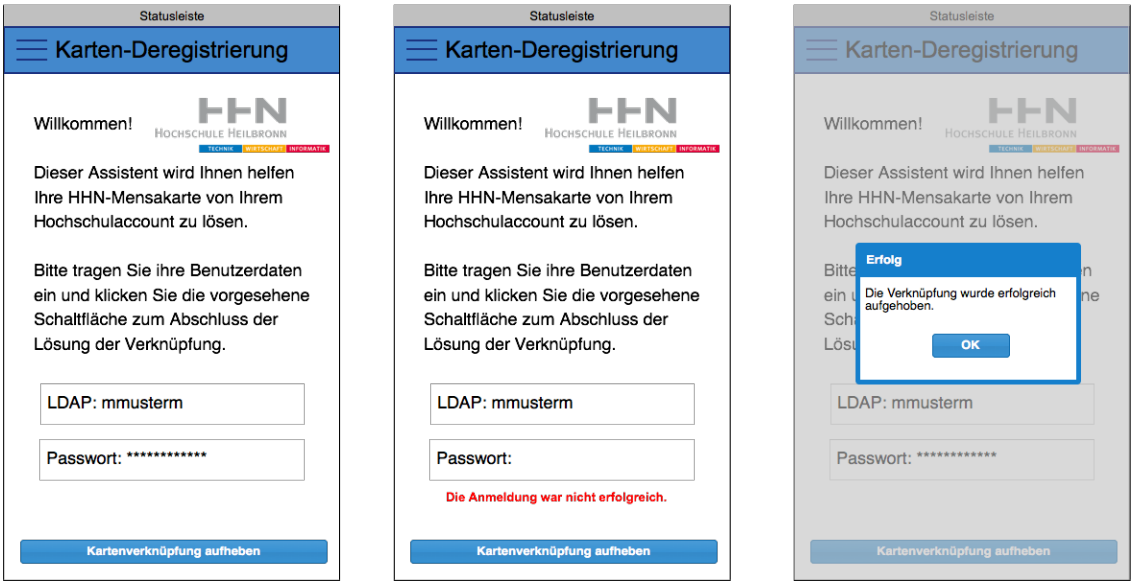

Abbildung 3.8: Papierbasierter Prototyp der Deregistrierung: Es werden der Benutzername und das Passwort des Benutzers benötigt. Ist die Verknüpfung erfolgreich, oder sind die Anmeldedaten ungültig, wird eine geeignete Meldung angezeigt.

4 Technologieevaluation

In diesem Kapitel werden nach den Anforderungen und Restriktionen aus Kapitel 3 Kriterien für Frameworks für hybride Applikationen und Implementierungen des JAX-RS 2.0-Standards aufgestellt, anschließend an den am Markt vorhandenen Lösungen evaluiert. Als Ergebnis dieser Analyse wird jeweils ein Framework für die Realisierung des Teilsystems ausgewählt. In Abschnitt 4.3 werden verschiedene Möglichkeiten der Versionierung von REST-Services vorgestellt und deren Vor- und Nachteile diskutiert.

4.1 Frameworks für hybride Apps

Die nachfolgenden Anforderungen ergeben sich aus den Anforderungen und Restriktionen (Abschnitt 3.3 und 3.4) für Frameworks hybrider Apps.

4.1.1 Formulierung Muss-Kriterien

Die folgenden Kriterien müssen von einer potentiellen Lösung vollständig erfüllt werden. Die Nichterfüllung eines Kriteriums führt zum Ausschluss der Lösung aus der weiteren Analyse.

- **Kostenmodell:** Das Framework muss vollständig kostenfrei verfügbar sein. Dies gilt auch für die Dokumentation und gegebenenfalls verfügbare Tutorials. Es darf auch nach der Entwicklung keine Folgekosten nach sich ziehen.
- **Lizenzmodell:** Die Software muss vollständig quelloffen sein und auf Open-Source-Software[1] basieren.
- **Plattform:** Die Android-Plattform muss ab Version 4.4 aufwärts unterstützt werden.
- **NFC:** Zugang zum NFC-Sensor des Smartphones muss über das Framework direkt oder ein entsprechendes Plugin zugesichert werden.
- **Release:** Das Framework soll auf neueren Technologien basieren und bis zur Implementierung der Anwendung im Mai 2015 ein stabiles Release vorweisen können.

[1] Die einzelnen Lizenzen können unter `http://opensource.org` eingesehen werden.

4.1.2 Auswertung Muss-Kriterien

Muss-Kriterien	**NativeScript**	**Ionic**	**Xamarin**
Kosten	Keine	Keine	$999/a
Lizenz	Apache 2.0	MIT	LGPLv2
Android	Ja	Ja	Ja
NFC	Nein	Ja[2]	Ja
Release, Zukunft	Juni 2015	Stabil	Stabil
Muss-Kriterien	**Famo.us**	**Kendo UI**	**AppGyver**
Kosten	Keine	$699/a	$199/a
Lizenz	MPLv2	Kommerziell[3]	Kommerziell
Android	Nein[4]	Ja	Ja
NFC	Ja[2]	Ja[5]	Ja[2]
Release, Zukunft	Stabil	Stabil	Stabil
Muss-Kriterien	**Trigger.IO**	**FireMonkey**	**Onsen UI**
Kosten	$79/m	Auf Anfrage	keine
Lizenz	Kommerziell	Kommerziell	Apache 2.0
Android	Ja	Ja	Ja
NFC	Nein	Ja[6]	Ja[2]
Release, Zukunft	Stabil	Stabil	Stabil
Muss-Kriterien	**Intel XDK**	**Sencha Touch**	**Appcelerator**
Kosten	keine	$3855/a	$39/m
Lizenz	Eigene	Kommerziell	Kommerziell
Android	Ja	Ja	Ja
NFC	Ja[2]	Ja	Nein
Release, Zukunft	Stabil	Stabil	Stabil
Muss-Kriterien	**Standardtechnologien[7]**		
Kosten	keine		
Lizenz	Phonegap: Apache 2.0, AngularJS, JQueryMobile: MIT		
Android	Ja		
NFC	Ja[2]		
Release, Zukunft	Stabil		

[2] Mit Cordova Plugin: `https://github.com/chariotsolutions/phonegap-nfc`
[3] Kern-Bibliothek mit Apache 2.0, Rest: `http://www.telerik.com/purchase/license-agreement/kendo-ui-professional`
[4] Framework ist primär für Websites gedacht, nur durch Kombination mit Phonegap/Cordova
[5] Über Plugin: `http://plugins.telerik.com/plugin/nfc`
[6] Durch Community-Plugin: `http://stackoverflow.com/questions/24650883/firemonkey-android-nfc-adapter`
[7] Phonegap/Cordova, AngularJS, JQuery

Auswertung der Muss-Kriterien:

Durch die Nichterfüllung von Kriterien und die Disqualifikation der Frameworks, werden in den weiteren Abschnitten die Frameworks Ionic (Abschnitt 2.7.3), Onsen UI (Abschnitt 2.7.6), Intel XDK (Abschnitt 2.6.5) und eine Komposition aus Standardtechnologien (Cordova - 2.6.2, AngularJS - 2.6.1, JQuery Mobile - 2.7.4) mit den weiteren Kriterien evaluiert.

4.1.3 Formulierung Soll-Kriterien

Die folgenden Kriterien sollten von einer potentiellen Lösung erfüllt werden um eine schnelle und effiziente Entwicklung und Wartung zu ermöglichen.

- Homogenes Framework: Geringe Fragmentierung der benutzten Frameworks für möglichst geringen Wartungsaufwand.
- Verfügbarkeit von kostenlosen Einstiegs-Tutorials und -Dokumentation.
- Freie Wahl der Entwicklungsumgebung, d.h. keine Bindung an frameworkspezifische Werkzeuge. Zudem sollten geringe Systemvoraussetzungen an Software und Hardware bestehen.
- Erweiterte Tools zur einfachen Testbarkeit und Entwicklung der App direkt auf dem Gerät oder im Browser. Cordova bietet bereits Funktionalität zum Test der App auf Gerät oder Emulator.
- Verbreitung: Bereits auf dem Markt etablierte Frameworks, finden in vielen Lösungen Anwendung. Durch ihre hohe Verbreitung sind Weiterentwicklung und kontinuierliche Beseitigung von Fehlern zugesichert.
- Erweiterbarkeit und Wartung des Frameworks: In das Framework sollen möglichst regelmäßig neue Funktionen integriert werden. Die Aktualisierung sollte mit möglichst wenig Aufwand durchführbar sein.

4.1.4 Auswertung Soll-Kriterien

Soll-Kriterien	Ionic	Onsen UI
Homogenität des Frameworks	Kapselung aller genutzten Technologien (Cordova, AngularJS, UI). Bereitstellung der Funktionalität der APIs über Command Line Interface (CLI).	Bereitstellung des reinen UI-Frameworks. Manuelle Verknüpfung mit Phongap/Cordova und AngularJS nötig.
Tutorials/ Dokumentation (kostenlos)	Einstiegsdokumentation für Installation, Dokumente für CLI, Entwicklung bis Ausrollen in App-Store beschrieben. Grobe Dokumentation über CSS-Styles des Frameworks. Verweis auf die AngularJS-Dokumentation. FAQ und offizielle Community vorhanden.	Detailierte Dokumentation der UI-Komponenten, detaillierte Beschreibung der einzelnen HTML-Tags und CSS-Styles. Installation und Einrichtung des Frameworks sehr kurz beschrieben. Keine offizielle Community vorhanden.
Bindung an Werkzeuge/ Systemvoraussetzungen	Keine Bindung an spezielle Werkzeuge. Ionic wird als NodeJS-Modul mit Cordova und AngularJS automatisch bei der Installation heruntergeladen und eingerichtet.	Eigenes Werkzeug zur Entwicklung verfügbar, keine Bindung an ein Werkzeug. Wird als NodeJS-Modul heruntergeladen. Manuelle Installation.
Testbarkeit der App direkt auf Gerät/Emulator	Die App kann während der Entwicklung direkt im Browser gestartet werden. Änderungen werden live übernommen.	Keine eigenen Hilfestellungen vorhanden.
Basierend auf verbreiteten Technologien	Ionic UI enthält Elemente von JQuery Mobile	Basiert auf keinen anderen Technologien.
Erweiterbarkeit und Wartung	AngularJS: Version 2 erschienen, Ionic plant die Integration. Ionic: ständiges Release von Features, Update durch CLI-Befehl automatisch.	Keine Informationen über UI-Framework selbst. Manuelle Suche nach neuen Versionen und Aktualisierung.

Soll-Kriterien	Intel XDK	Standardtechnologien
Homogenität des Frameworks	Entwicklungsumgebung kapselt Phonegap/Cordova.	Die einzelnen APIs werden manuell miteinander verbunden.
Tutorials/ Dokumentation (kostenlos)	Keine Tutorials zur Entwicklung mit HMTL5. Dokumentation zur Installation und Inbetriebnahme der Entwicklungsumgebung.	Tutorials und Dokumentation der einzelnen Technologien. Community für alle Technologien vorhanden.
Bindung an Werkzeuge/ Systemvoraussetzungen	Eine Entwicklung mit anderen Werkzeugen als Intel XDK scheint nicht vorgesehen zu sein.	Freie Wahl der Entwicklungsumgebung.
Testbarkeit der App direkt auf Gerät/Emulator	Das Integrated Development Environment (IDE) bietet Möglichkeiten die App visuell mit den Cordova-Mitteln zu Testen. Eine Entwicklung mit „Live Layout Editing“ der Seiten ist möglich.	Debugging über verwendete Entwicklungsumgebung oder mit Browser-Entwicklertools möglich.
Basierend auf verbreiteten Technologien	Kapselt Phonegap/Cordova.	Repräsentieren die Standardtechnologien des Markts
Erweiterbarkeit und Wartung	Entwicklungsumgebung: Updates als einziges Mittel. Ausreichend Funktionalität für unerfahrene Benutzer. Fortgeschrittene vermissen schnell weitere Funktionen wie z.B. Code-Vervollständigung.	Wartung der einzelnen Frameworks sehr einfach. Regelmäßige AngularJS- und JQuery mobile -Releases vorhanden.

4.1.5 Ergebnis

Intel XDK (Abschnitt 2.6.5) scheint für Einsteiger durch Vorgabe der Entwicklungsumgebung viele hilfreiche Funktionen zu besitzen und erleichtert durch die grafische Benutzeroberfläche den Einstieg in die Entwicklung hybrider Applikationen. Erfahrenere Benutzer vermissen schnell weitere Funktionalität und zusätzliche Dokumentation.

Onsen UI (Abschnitt 2.7.6) besticht durch die äußerst detaillierte Dokumentation. Informationen zu regelmäßigen Updates und Funktionserweiterungen sind nicht verfügbar. Potenziell vorhandene Vorteile gegenüber anderen Frameworks wurden nicht evaluiert, da die Geschwindigkeit des Seitenaufbaus, die Vielfalt an Oberflächenkomponenten und Animationen keine für die Applikation essentiellen Faktoren sind.

Das Aggregat der Standardtechnologien bietet durch deren breite Online-Community ausreichende Hilfestellung für Einrichtung und Entwicklung einer hybriden Applikation. Eine effiziente Entwicklung benötigt jedoch weitere unterstützende Hilfsmittel und gegebenenfalls zusammenfassende Literatur, da der Umfang der Dokumentation teilweise schwer überschaubar ist oder Dokumente veraltet sind.

Ionic Framework (Abschnitt 2.7.3) bietet durch die Kapselung von Standardtechnologien, die umfangreiche Dokumentation, viele nützliche Entwicklertools und regelmäßige Aktualisierungen die reifeste Hilfestellung aller evaluierten Frameworks. Diese ist sowohl für Einsteiger, als auch für fortgeschrittene Entwickler geeignet und wurde für die Realisierung der hybriden Applikation ausgewählt.

4.2 JAX-RS Implementierungen & JSON-Umwandlung

Der JAX-RS 2.0-Standard sieht alle für einen Entwurf mit Trennung von Service-Definition und -Implementierung nötigen Hilfsmittel, wie z.B. Annotationen vor (siehe Abschnitt 2.3). Die Implementierungen des Standards unterscheiden sich in der Hilfestellung der Konvertierung von Objekte zu Formaten wie der Extensible Markup Language (XML) oder JSON, ihrer Einrichtung und Funktionalität über den Standard hinaus.

Es gibt mehrere Verfahren, Objekte in sprachunabhängige Formate wie z.B. XML und JSON zu konvertieren, von denen die drei evaluiert werden. Für das Projekt ist besonders das Datenformat JSON auf Grund der mit Webstandards zu entwickelnden hybriden Applikation relevant.

In den folgenden Abschnitten werden die Vor- und Nachteile der manuellen (Abschnitt 4.2.1) und automatischen Umwandlung von Objekten mittels Provider (Abschnitt 4.2.2) sowie Java Architecture for XML Binding (JAXB) (Abschnitt 4.2.3) analysiert. Alternativ zur Verwendung eines Frameworks wird in Abschnitt 4.2.4 die Lösung des Services über ein Servlet vorgestellt.

4.2.1 Manuelle Umwandlung mit Hilfsklassen

Die in Listing 4.1 abgebildete Klasse enthält eine Methode für die JSON-Übersetzung. Die verwendeten Hilfsklassen sind unter `http://www.json.org/java/index.html` erhältlich. Felder komplexer Typen, sowie Arrays und Collections müssen manuell kon-

vertiert, Objektreferenzen auf `null` getestet und rekursiv in JSON umgewandelt werden.

```
1  import org.json.*;
2
3  public class Person {
4    private int id;
5    private String name = "";
6      private List<Person> children = new ArrayList<Person>();
7
8    // getter + setter
9
10   public String getJSON() {
11     JSONObject dataset = new JSONObject();
12     dataset.put("name", name);
13     dataset.put("id", id);
14     JSONArray list = new JSONArray();
15     for(Person p : children){
16         list.add(p.getJSON());
17     }
18     dataset.put("children", list);
19     return dataset.toString();
20   }
21 }
```

Listing 4.1: Beispiel für die manuelle Umwandlung von Objekten zu JSON. In den Zeilen 15ff befindet sich eine bei zyklischen Abhängigkeiten entstehende Endlosschleife.

Bezüglich der Erweiterbarkeit und Wartung des Codes sind Einschränkungen gegeben. Die Methode muss bei Rückgabe ein Objekt vom Typ String zurückgeben. Der Service muss die Umwandlung des Objekts zu seiner JSON-Repräsentation aktiv durchführen. Eine Auslagerung der Übersetzung in den Service sorgt für eine engere Kopplung zwischen Service und Datentransferobjekt (DTO), besitzt aber den Vorteil, dass der Service nur die benötigten Felder des Objekts an Stelle einer universellen Repräsentation übersetzen muss. Zusätzlich ist zu beachten, dass beim Empfang von JSON der einkommende String aufbereitet und in ein Objekt umgewandelt werden muss, was ebenfalls aktiv durchgeführt werden muss.

4.2.2 Automatische Umwandlung mit Provider

Mit Providern, einer Schnittstelle für die Umwandlung beliebiger Medientypen, die der Standard in Form von Plug-Ins vorsieht, werden Objekte automatisch in JSON konvertiert, ohne das diese annotiert sein müssen. Die Implementierung eines solchen Providers ist jedoch nicht vom Standard abgedeckt, sodass für jedes Framework ein Provider vorhanden sein muss, um nicht selbst einen solchen schreiben zu müssen.

Der am weitesten verbreitete Provider für die JSON-Umwandlung ist Jackson. Provider werden mit `javax.ws.rs.ext.Provider` annotiert und beim Start der Applikation automatisch geladen, d.h. es ist keine weitere Einrichtung nötig, sofern nicht mehrere Provider für das selbe Datenformat oder unterschiedliche Datenformat-Schemata verwendet sollen. Die Schnittstelle kann dadurch als Rückgabe den Typ des Objekts besitzen. Listing 2.2 in Abschnitt 2.3 zeigt einen Service, der Objekte der Klasse `Person` automatisch über einen Provider in JSON übersetzt.

4.2.3 Automatische Umwandlung über JAXB

Ein zu serialisierendes Objekt kann mit JAXB in beiden Richtungen in JSON und XML umgewandelt werden. Klassen und Felder werden mit den in JAXB üblichen Annotationen zum Mapping versehen. Die Verbindung zwischen JAX-RS und JAXB wird durch einen Provider hergestellt. Das Mapping zu XML und JSON kann durch explizite Angabe eines Schemas in der Klasse verändert werden. Alternativ können JAXB-Mapper geschrieben werden, die das Mapping von Objekt zu XML festlegen.

```
1 import javax.xml.bind.annotation.XmlElement;
2 import javax.xml.bind.annotation.XmlRootElement;
3
4 @XmlRootElement
5 public class Person {
6
7   private int id=0;
8   private String name="";
9
10  @XmlElement
11    public int getId() { return id; }
12  public void setId(int id) { this.id = id;}
13
14    @XmlElement
15  public String getName() { return name; }
16  public void setName(String name) { this.name = name; }
17 }
```

Listing 4.2: Zu serialisierende Objekte müssen mit den JAXB-Annotationen versehen werden. Der Provider wandelt das Ergebnis/ die Eingabe über JAXB automatisch um.

```
1 {"person":{"id":1,"name":"JAXB"}} # Umwandlung mit JAXB
2 {"id":1,"name":"Provider"} # Umwandlung mit Provider
```

Listing 4.3: Das Ergebnis unterscheidet sich in seiner Form. Bei der Umwandlung mit JAX-B muss ein Wurzelelement vorhanden sein.

4.2.4 Alternativlösung mit Servlets

Bei diesem Ansatz wird kein Framework für die REST-Schnittstelle benutzt. Ein Servlet wartet auf Anfragen, delegiert die Objekterzeugung an eigene Service-Klassen, und gibt die Ergebnisse zurück. Dieser Ansatz ist für den Entwickler sehr aufwändig, da bei komplexen Anfragen alle Parameter aus der Uniform Resource Locator (URL) extrahiert, encodiert und z.B. nach Typ und Wertebereich validiert werden müssen, bevor eine Verarbeitung der Anfrage möglich ist.

```
1  import java.io.*;
2  import javax.servlet.ServletException;
3  import javax.servlet.http.*;
4
5  public class ServiceServlet extends HttpServlet {
6
7      protected void doGet(HttpServletRequest request,
8        HttpServletResponse response) throws ServletException,
             IOException {
9      PrintWriter out = response.getWriter();
10     String url = request.getPathInfo();
11     int id = 0;
12     String name = "no name specified";
13     if (url.startsWith("/person/")) {
14       url = url.substring("/person/".length(), url.length());
15       try {
16         id = Integer.parseInt(url);
17       } catch (Exception e) {
18         out.println("id not valid");
19       }
20       if (request.getParameterMap().containsKey("name")) {
21         name = request.getParameter("name");
22       }
23     }
24     out.println(PersonService.getPerson(id, name));
25     out.close();
26   }
27 }
```

Listing 4.4: Beispiel-Servlet

Es ist zu beachten, dass das Servlet in der `web.xml` der Anwendung registriert und auf den Basispfad gemappt werden muss. Es ist daher sinnvoll für die einzelnen Services und deren Versionen jeweils ein eigenes Servlet zu implementieren. Diese werden dann auf verschiedene Pfade registriert und können in der `web.xml` gezielt aktiviert und deaktiviert werden, um Schnittstellen zu deaktivieren und neue hinzuzufügen.

4.2.5 Ergebnis

Für die Entwicklung des REST-Services wird die JAX-RS-Implementierung RESTEasy verwendet. Durch die Vielzahl an verfügbaren Providern für Datenformate und umfassende Contexts and Dependency Injection (CDI) zur Registrierung von Services und der Einrichtung des Frameworks minimiert sich der Entwicklungsaufwand mit RESTEasy erheblich. Zwar sind das Restlet Framework und Apache CXF bezüglich des Funktionsumfangs mächtiger als RESTEasy oder Jersey, jedoch ist die Einarbeitung und Einrichtung beider Frameworks aufwändiger. Jersey, die Referenzimplementierung des Standards bietet eine Alternative zu RESTEasy, jedoch sind nicht für alle Datenformate entsprechend flexible Provider verfügbar. Die Auswahl erfolgt nach Implementierung eines Test-Services unter Verwendung von RESTEasy und Jersey, sowie den verschiedenen Möglichkeiten zur JSON-Objekt-Umwandlung.

Für die Umwandlung von Java-Objekten in JSON und umgekehrt wurde das Verfahren über JAXB ausgewählt. Durch das Annotieren der zu übersetzenden Felder besitzt der Ansatz die Flexibilität bezüglich multipler Repräsentationen des gleichen Objekts oder Schemaänderungen der Ausgabe. Die manuelle Übersetzung mit Hilfsklassen besitzt eine ähnliche Flexibilität, der initiale Aufwand für die Entwicklung der umkehrbaren Übersetzung ist jedoch höher. Provider bilden einen Mittelweg zwischen manueller und kontrollierter automatischer Übersetzung mit JAXB. Bei Anpassung von Schemata oder partieller Übersetzung von Objekten, steigt jedoch auch hier der Aufwand.

4.3 Versionierung von App und Server-Backend

In diesem Abschnitt werden verschiedene Möglichkeiten der Versionierung von REST-Services vorgestellt.

REST-Services brauchen nicht aktiv versioniert werden, da in der Regel nur die Ressourcen veränderlich sind, bzw. bei Erweiterungen neue Services implementiert oder weggelassen werden. Häufig werden jedoch einfache Remote Procedure Calls (RPCs) über HTTP durchgeführt. Diese ähneln stark den Prinzipien von REST und müssen versioniert werden. Diese sind die Codierung in die URL (Abschnitt 4.3.1) und die Codierung in den Medientyp (Abschnitt 4.3.2).

Eine Versionierung der Smartphone-App für Android wird nicht durchgeführt, da bei der Veröffentlichung die letzte Version benötigt wird. Ältere Versionen können durch bei der Entwicklung verwendete Versionskontrollwerkzeuge wiederhergestellt werden. Eine Benennung der Versionen wird nach den Google Richtlinien ([27]) durchgeführt.

4.3.1 URL-Codierung

Ein weit verbreiterter Ansatz zur Versionierung von Services ist die Codierung der Versionsnummer in die URL. Pro URL können die verschiedenen Medientypen, wie z.B. JSON, XML, Text, konsumiert und produziert werden. Durch die URL-Versionierung muss für jede neue Service-Version unter Umständen der komplette Service-Baum dupliziert werden. Ist die URL nicht mehr gültig, weil die ältere Version eines Service deaktiviert wurde, so wird vom Server der HTTP-Statuscode `404 (Not found)` zurückgegeben, da es keine entsprechende Ressource gibt.

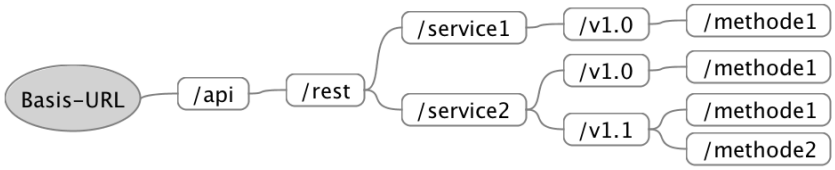

Abbildung 4.1: Versionierung eines REST-Service über die URL.

Es ist ersichtlich, dass die Versionierung nicht eindeutig erfolgen muss, was das Warten der Schnittstelle aufwändiger gestaltet. Der URL-Baum kann sich an jedem beliebigen Punkt verästeln, was weitere Komplexität und Code-Duplizierung zur Folge hat. Als Konsequenz sollte die Schnittstelle so konzipiert werden, dass sie Vertrag der Kommunikation zwischen Client und Server ist, und damit nur wenige Änderungen erfährt.

4.3.2 Media-Type-Codierung

Ein junger Ansatz der Versionierung des Service bietet die Codierung in den Medientypen, wie z.B. JSON, XML, Text. Für diese sind entsprechende Provider zum komfortablen Mapping vorhanden. Zusätzlich können eigene Medientypen erstellt werden, die bereits bestehende erweitern.
Bsp.: `application/json` wird zu `application/abc+json` erweitert.

Der Vorteil dieses Ansatzes ist, dass alle Anfragen an die selbe URL erfolgen. Der Medientyp wird im Header der Anfrage mitgesendet, so können auch mehrere Typen mitangegeben werden, und Präferenzen mitgeteilt werden. Dieses Verhandeln des Medientyps zwischen Client und Server heißt *Content Negotiation*. Es kann somit über das zugrundeliegende Datenformat zwischen den Versionen der REST-Schnittstelle unterschieden werden, was zum einen die Unterstützung für ältere Clients sichert, zum anderen die Schnittstelle flexibel hält. Unterstützt der Server den Medientyp nicht, so wird ein HTTP-Statuscode `415 (Unsupported media type)` zurückgegeben.

Einige Technologien und Standards unterstützen diesen Ansatz bereits, so auch JAX-RS. Eine Unterscheidung wird dabei über die Annotation `javax.ws.rs.Consumes` an den Service-Methoden vorgenommen.

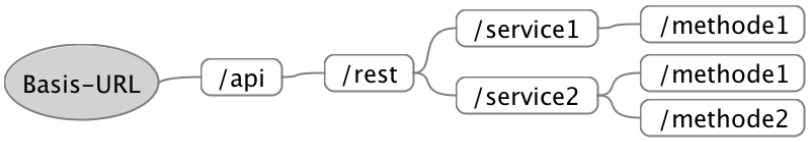

Abbildung 4.2: Die Versionierung der Schnittstelle hat keinen Einfluss auf die Pfade der Ressourcen und Services.

4.3.3 Ergebnis

Die Versionierung durch den Medientypen ermöglicht das Aushandeln der verwendeten Schnittstellen-Version, was die Migration der Clients vereinfacht. Als Framework für die hybride Applikation wird nach Auswahl (Abschnitt 4.1.5) Ionic Framework in Verbindung mit AngularJS verwendet. Dieses unterstützt in der jetzt verfügbaren Version nur die unveränderlichen Medientypen. Daher wird für die Versionierung des REST-Services die URL-Codierung verwendet. Die Komplexität der Schnittstelle ist für den Entwickler zwar höher als bei einer Codierung über den Medientypen, kann durch ein entsprechendes Konzept jedoch gering gehalten werden.

5 Entwurf

In diesem Kapitel ist der Entwurf des verteilten Systems dargestellt. In Abschnitt 5.1 wird zunächst auf die Verteilung des Systems auf die verschiedenen Geräte und Netze eingegangen. In den Abschnitten 5.2 und 5.3 wird die Architektur des Services und der hybriden App dargestellt und erläutert.

5.1 Systemarchitektur

Das System besteht aus der hybriden Applikation, die auf den Smartphones der Nutzer installiert ist, und einem REST-Service, der auf einem Webserver der Hochschule Heilbronn auf Anfragen der Clients wartet. Die App kommuniziert über eine sichere Verbindung mit dem Service. Dieser behandelt alle Anfragen der Clients zur Registrierung und Deregistrierung. Um Änderungen an der Datenbasis vornehmen zu können erhält die Applikation nach Abschluss der Arbeit Zugriff auf das Personenverzeichnis der Hochschule Heilbronn. Für die Dauer der Arbeit wird eine Datenbank für Tests eingerichtet.

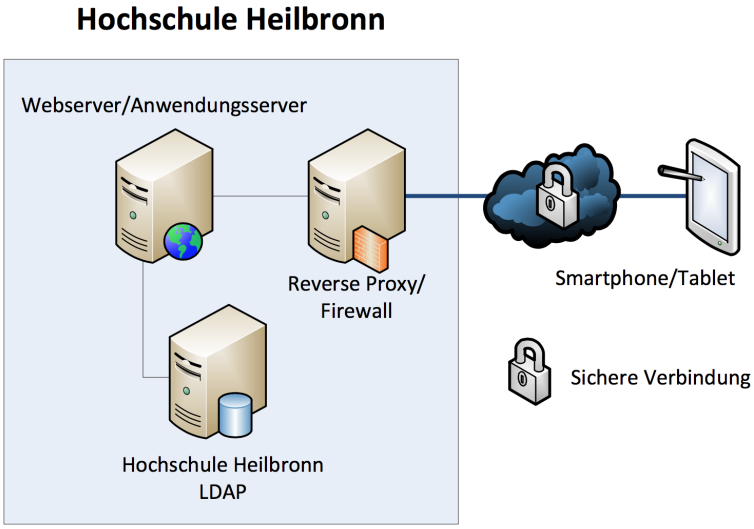

Abbildung 5.1: Beispielhafte Verteilung der Systemkomponenten im Netzwerk. Die Übertragung über unsichere Netze ist immer verschlüsselt. Interne Netze sind durch einen Rahmen gekennzeichnet.

5.2 Architektur des Server-Backends

Die Architektur des Server-Backends gliedert sich in drei Schichten (vgl. Abbildung 5.2). Die erste Schicht bildet die Zugriffsschicht auf die Anwendung und wird durch den REST-Service repräsentiert. Dieser nimmt die Anfragen der Clients an und wandelt die Eingabe, sowie die Antworten des darunterliegenden Services in den erwarteten Datentyp um. Die Schicht repräsentiert daher den zustandslosen Stellvertreter des objektorientierten Service. Die direkt darunterliegende Schicht ist der auf die Datenhaltung zugreifende Service. Der Service beinhaltet die Logik, die notwendig ist, um die Anfragen der darüberliegenden Schicht zu behandeln und konsistente und valide Zustände in der Datenhaltung zu gewährleisten.

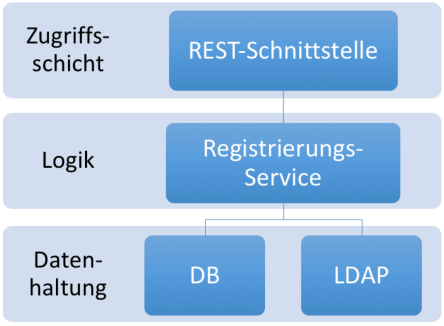

Abbildung 5.2: Schichten des Server-Backends.

Um den REST-Service nach dem in Abschnitt 4.3.1 vorgestellten Prinzip über die URL versionieren zu können, werden grundlegende Funktionen der Schnittstelle im Interface `TagService` festgehalten (siehe Abbildung 5.3). Diese hat keine Abhängigkeiten zu JAX-RS, definiert aber die Eingabe und Ausgabe.

Da das Verhalten aller Services bezüglich der Aktualisierung der Verknüpfung gleich ist, wird die gemeinsame Funktionalität in die Klasse `DelegateTagService` ausgelagert. Unter deren Benutzung muss ein REST-Service lediglich die Konvertierung von Eingabe und Ausgabe vornehmen und die Durchführung der Aktualisierung delegieren. Soll der Ablauf erweitert werden, kann dies über die vorgesehenen Einschubmethoden erfolgen.

Das Interface `NfcService` der Service-Schicht definiert die Schnittstelle zwischen konkreten, die Datenbasis verändernden Implementierungen der Funktionalität und der Zugriffsschicht. Eine konkrete Instanz zum Service dieser Schicht, wird dem Objekt der Klasse `DelegateTagService` bei der Instantiierung mit der `NfcServiceFactory` injiziert.

Die zentralisierte Instantiierung ermöglicht die Konfiguration und Austauschbarkeit der verwendeten Services und reduziert die Abhängigkeiten zwischen den Schichten. Im Diagramm ist angedeutet, dass für unterschiedliche Datenhaltungen oder Zugriffsarten darauf jeweils konkrete `NfcServices` implementiert werden, die die Abhängigkeiten zur Datenhaltung auflösen. Für die Arbeit wurde beispielhaft der Zugriff auf eine relationale Datenbank über Java Persistence API (JPA) und einen Object/Relational-Mapper (ORM) realisiert.

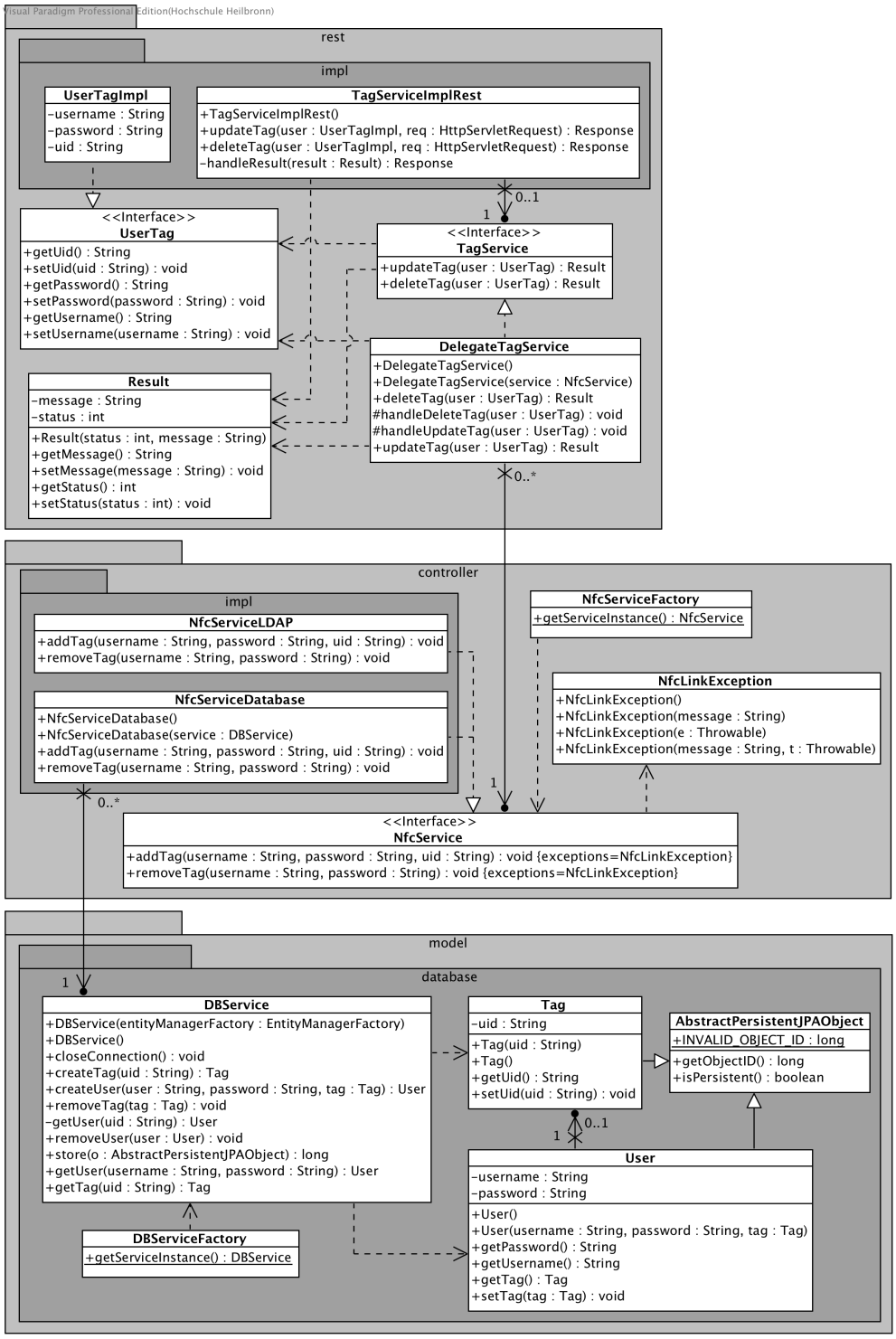

Abbildung 5.3: Klassendiagramm: Die Architektur des Server-Backends gliedert sich in Zugriffsschicht, Service und Datenhaltung.

5.3 Architektur der hybriden App

Die Architektur der hybriden Applikation mit Phonegap und AngularJS wurde in den Abschnitten 2.6.2 und 2.6.1 allgemein beschrieben. Dieser Abschnitt erläutert den Aufbau der Web-App und Verwaltung der Ressourcen (Abschnitt 5.3.1) für eine Erleichterung der Integration weiterer Module, sowie die verwendeten Cordova-Plugins (Abschnitt 5.3.2).

5.3.1 Modularisierung und Erweiterbarkeit

In Ionic-Projekten ist initial die Sortierung der Ressourcen, wie z.B. JavaScript-, HTML- und CSS-Dateien, nach Dateityp eingerichtet. AngularJS ermöglicht die Trennung Modul-Definition von Controllern, Konstanten und Navigation. Als Folge werden diese im Projekt, nach Modul-Funktionalität sortiert, in einzelne Dateien ausgelagert und in einem modulspezifischen Verzeichnis abgelegt. Diese Separation of Concerns (SoC) ist in Abbildung 5.4 veranschaulicht und vereinfacht die Integration weiterer Module.

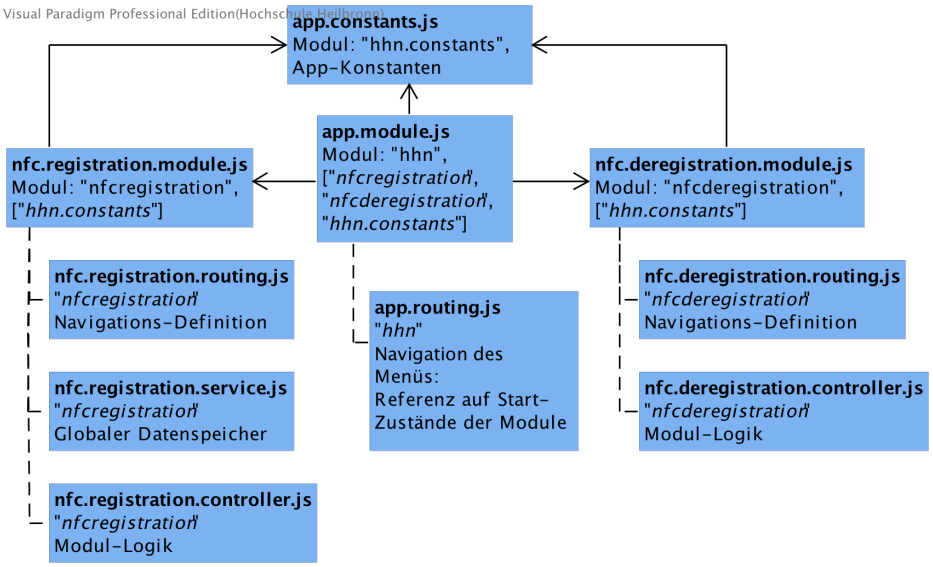

Abbildung 5.4: Modellierung der Projektstruktur: Die Trennung der Moduldefinition von Navigation, Controllern, Konstanten ist als gestrichelte Linie visualisiert. Abhängigkeiten sind als gerichtete Kanten dargestellt.

In Controller-Instanzen erzeugte Variablen sind nur in dessen Sichtbarkeitsbereich, *Scope* genannt, zugänglich. Beim Seitenwechsel wird der aktuelle Scope verworfen und eine neue Instanz des Controllers erzeugt. Als Folge kann eine Seite des Registrierungsassistenten nicht auf die Variablen vorheriger Seiten zugreifen. Um dennoch Felder austauschen zu können, wird ein modulweit verfügbarer Datenspeicher definiert

(vgl. `nfc.registration.service.js`), in welchen die Seiten Variablen belegen und abfragen können. Dieser Speicher wird bei der Erzeugung eines Controllers mit den benötigten Konstanten eines Moduls injiziert.

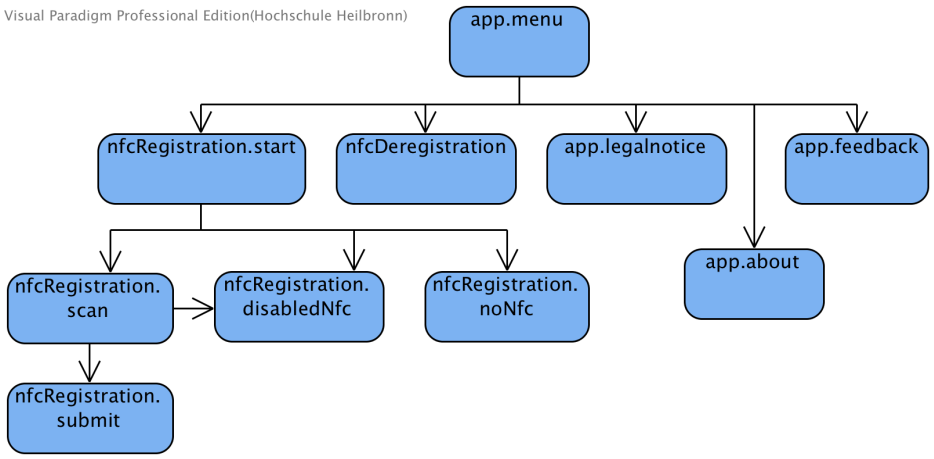

Abbildung 5.5: Alle Modul-Zustände sind über das Menü erreichbar.

Die Navigation besteht aus Zuständen, die ein Triplet aus einer HTML-Seite zur Darstellung, einer URL und einem eindeutigen Namen für den Zugriff bilden. Jeder Zustand eines Moduls muss durch Transitionen anderer Zustände erreichbar sein. Durch eine dezentralisierte Navigation können modulspezifische Navigationen den Navigationsautomaten beliebig aber eindeutig erweitern. Transitionen zwischen Zuständen verschiedener Module sollten nicht vorhanden sein, um zusätzliche Abhängigkeiten zwischen eigenständigen Modulen zu vermeiden. Das Zusammenführen aller Modul-Zustände erfolgt im Menü der Applikation (vgl. Abbildung 5.5).

5.3.2 Cordova-Plugins

Durch eine Plugin-Architektur ermöglicht Cordova die Integration und Austauschbarkeit von Plugins für die Erweiterung des Zugriffs auf Gerätefunktionen.

Für die Nutzung des NFC-Sensors wird ein auf GitHub verfügbares Plugin[1] verwendet, das die Verfügbarkeit des Sensors, über die reine Benutzung hinaus, testen kann.

Da, Cordova grundsätzlich den Zugriff auf fremde Ressourcen sperrt (siehe Abschnitt2.5) wird ein Whitelist-Plugin[2] für Cordova hinzugefügt, dass den Zugriff auf Ressourcen der angegebenen Quellen erlaubt. Dabei wird die CORS-Beschränkung gelockert.

[1] `https://github.com/chariotsolutions/phonegap-nfc`. Das NFC-Cordova-Plugin stammt von Don Coleman, der mit Tom Igoe Author eines Anfang 2014 erschienen Buches [28] über die Nutzung von NFC mit Arduino, Android und Phonegap ist.

[2] `https://github.com/apache/cordova-plugin-whitelist`

5.4 Kommunikation zwischen App und Server

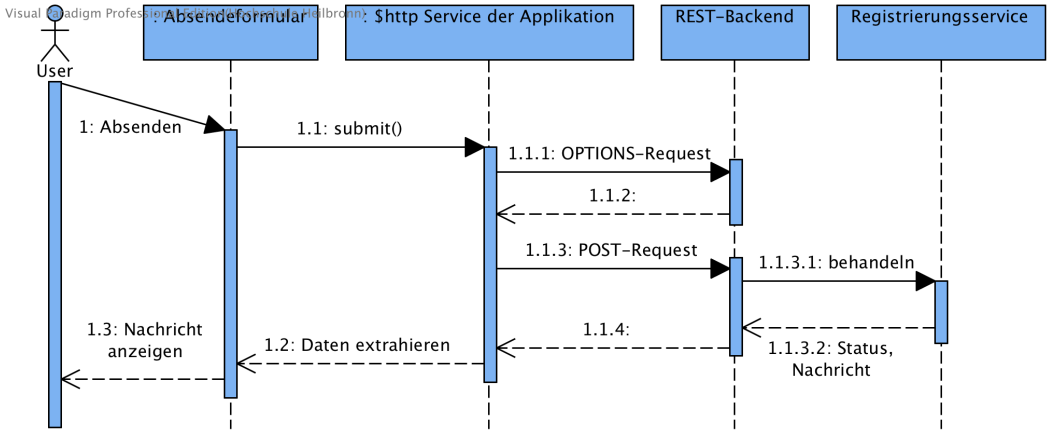

Abbildung 5.6: Nach der Initiierung des Absendens wird zunächst automatisch ein OPTIONS-Request gesendet, um CORS-Regeln abzufragen. Der nachfolgende POST-Request enthält die Benutzerdaten.

Die Applikation ist bis auf das Absenden der Benutzerdaten vollständig offline nutzbar. Alle Ressourcen, wie z.B. JavaScript- oder HTML-Dateien, sind in das Build-Artefakt integriert. Folglich kommunizieren App und Server nur beim Austausch der Benutzerdaten der Registrierung und Deregistrierung der NFC-Karten. Das Absenden wird durch den Benutzer initiiert und führt dazu, dass die Applikation einen POST-Request mit den Benutzerdaten, bei der Registrierung zusätzlich der Karten-ID, an den Server sendet (vgl. Listing 5.1). Die CORS-Mechanismen führen zu einem automatischen Absenden eines vorausgehenden OPTIONS-Requests an den Server. Dieser antwortet mit den zugelassenen HTTP-Methoden, Authentifizierungsmöglichkeiten und CORS-Headern um dem Client den Zugriff auf die Ressource zu signalisieren. Erst, wenn der Client die Erlaubnis des Servers erhält, sendet er den eigentlichen POST-Request. Der Server antwortet unabhängig vom Ergebnis der Aktion mit einer JSON-Nachricht. Diese beinhaltet einen Statuscode und eine textuelle Nachricht, die dem Benutzer angezeigt wird (vgl. Listing 5.2).

```
1 {'usertag': {'password': 'abc123','username': 'snoopy','uid': '
      11:22:33:44:55:66:77'}}
```

Listing 5.1: Die Benutzerdaten und ggf. Seriennummer der NFC-Karte werden an den Server gesendet.

```
1 {'result': {'status': 200,'message': 'Feedback-Message'}}
```

Listing 5.2: Die Antwort des Servers enthält einen Statuscode, sowie eine Nachricht für den Benutzer.

6 Implementierung und Ergebnisse

Auf Basis des in Kapitel 5 vorgestellten Entwurfs wurde ein funktionaler Prototyp für App und Backend erstellt. Die Applikation besitzt den vollen in Kapitel 3 definierten Funktionsumfang und kann im Hochschulnetz für die Registrierung und Deregistrierung von NFC-Karten eingesetzt werden, weil das Backend durch einen Build-Server kontinuierlich übersetzt und auf einem Server der Medizinischen Informatik betrieben wird.

6.1 Prototyp der hybriden App

Im Folgenden wird die Realisierung der Registrierung (Abschnitt 6.1.1) und Deregistrierung (Abschnitt 6.1.2) auf Seite der hybriden Applikation dargestellt.

6.1.1 Registrierung

Die Oberfläche des implementierten Prototyps der Startseite des Registrierungsassistenten mit Hinweisen zum benötigten NFC-Sensor ist in Abbildung 6.1 dargestellt. Der im Papierentwurf konzipierte modale Dialog während der Erfassung des Sensorstatus musste nicht umgesetzt werden, da der Sensorstatus durch das verwendete Plugin direkt von der Plattform ausgelesen werden konnte.

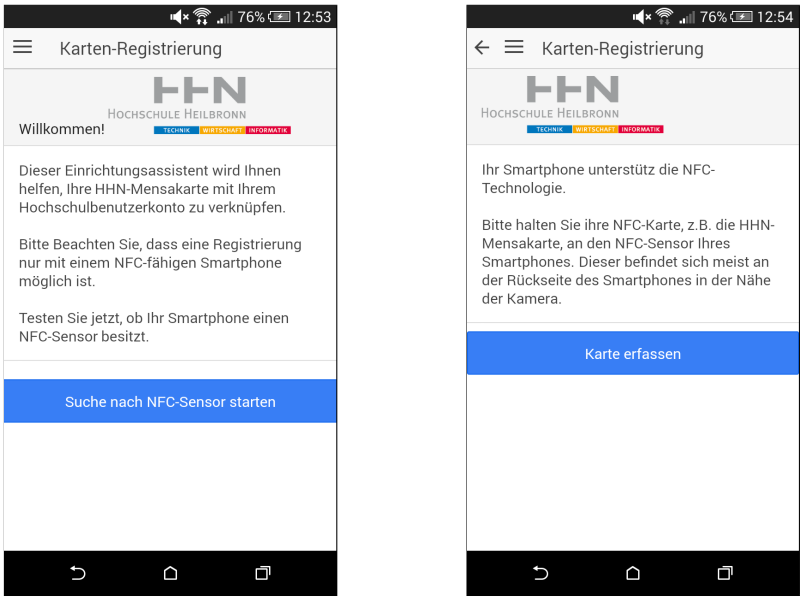

Abbildung 6.1: Startseite der Registrierung zum Test des Sensorstatus (links) und der Kartenerfassung (rechts).

In Listing 6.1 ist die Methode zum Testen des Sensorstatus dargestellt. Diese bewirkt eine Umleitung auf eine Fehlerseite, wenn der Sensor nicht aktiviert ist. Die Schnittstelle zum NFC-Plugin bildet das injizierte `nfc`-Objekt. Die Methode `enabled` dieses Objekts erwartet zwei Methoden, wobei die erste das Verhalten bei aktiviertem Sensor und die zweite das alternative Verhalten unter Berücksichtigung des Grundes spezifiziert. Ist der Sensor aktiviert, wird keine Aktion durchgeführt und keine Umleitung vorgenommen. Die Methode kann daher direkt vor der Nutzung des Sensors erneut aufgerufen werden, um eine zwischenzeitliche Statusänderung des Sensors erkennen zu können.

```
1 function checkSensor() {
2     nfc.enabled(
3         function () {},
4         function (reason) {
5             if (reason == 'NO_NFC') {
6                 $state.go('app.nfcRegistration-no');
7             } else if (reason == 'NFC_DISABLED') {
8                 $state.go('app.nfcRegistration-disabled');
9             } else {
10                $state.go('app.nfcRegistration');}
11        }
12    );
13 }
```

Listing 6.1: Die Methode nutzt das NFC-Plugin, um den Sensorstatus zu bestimmen. Alternative Zustände des Sensor führen zur Umleitung auf eine Fehler-Seite.

Durch das Betätigen des "Karte erfassen"-Buttons im rechten Bild von Abbildung 6.1 wird über das NFC-Plugin ein Listener am NFC-Sensor registriert. Listing 6.2 zeigt den Aufruf der entsprechenden Methode des `nfc`-Objekts. Dem Aufruf dieses Blocks geht eine erneute Verwendung der in Listing 6.1 vorgestellten Methode voraus. Die Methode erwartet drei Funktionen, die das Verhalten bei einer erkannten Karte, der erfolgreichen und erfolglosen Registrierung des Listeners spezifiziert.

Wird eine Karte erkannt, so wird mit der ersten Methode die Seriennummer der Karte ausgelesen und in eine lesbarere Hex-Repräsentation mit Trennzeichen[1] umgewandelt. Die so erhaltene Seriennummer wird im Service des Moduls abgelegt, um sie, nach dem Entfernen des Listeners und einer Weiterleitung zum Absendeformular, nutzen zu können (vgl. Abschnitt 5.3.1). Die zweite Methode führt bei erfolgreicher Registrierung zur Anzeige eines modalen Dialogs. Dieser ist in Abbildung 6.2 dargestellt.

[1] Die Bytes der Seriennummer werden durch einen Doppelpunkt getrennt. Bsp.: `11:22:33:44:55:66:77`

```
nfc.addTagDiscoveredListener(
    function (nfcEvent) {
        String uid = getFormattedUID(nfcEvent.tag.id);
        NfcRegistrationService.setUid(uid);
        removeListener();
        $scope.modal.hide();
        $state.go('app.nfcRegistration-submit');
    }, function () {
        $scope.modal.show();
    }, function (reason) {
        removeListener();
        $scope.modal.hide();}
);
```

Listing 6.2: Die Methode nutzt das NFC-Plugin, um die Seriennummer der Karte auszulesen. Nach einer Umwandlung wird diese in den modulweiten Datenspeicher abgelegt.

Die Oberfläche des implementierten Prototyps des modalen Dialogs zum Auslesen der Seriennummer und das Absendeformular ist in Abbildung 6.2 dargestellt. Der Dialog enthält Hinweise für die Registrierung. Der Benutzer initiiert die Verknüpfung durch das Auslösen von "Karte verknüpfen".

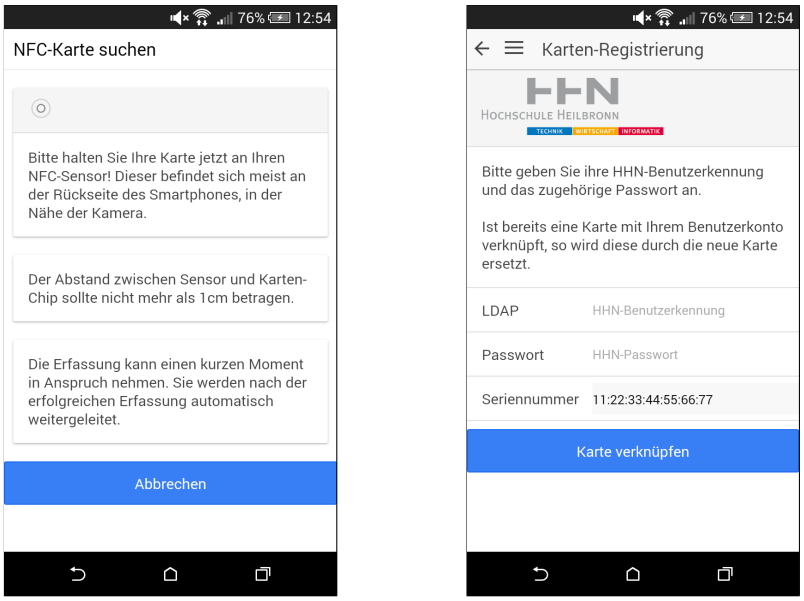

Abbildung 6.2: Modaler Dialog zum Auslesen der Karten-Seriennummer (links) und Absendeformular (rechts).

Nach dem Auslösen der Verknüpfung wird, wie in Abschnitt 5.4 beschrieben, eine Anfrage an den Server abgeschickt. Die erhaltene Antwort wird dem Benutzer nach Statuscode farbcodiert (siehe Abbildung 6.3) in rot (erfolglos) oder grün (erfolgreich) unter den Eingabefeldern des Formulars dargestellt (siehe Abbildung 6.4). Ist das Formular nicht vollständig ausgefüllt erfolgt äquivalent die Anzeige einer Meldung in gelb, eine Anfrage wird nicht getätigt.

Abbildung 6.3: Die Antwort des Servers wird dem Benutzer eine farblich hervorgehoben dargestellt.

Der Datenspeicher des Registrierungs-Moduls (siehe Listing 6.3) besitzt nur das Feld für die Seriennummer der gelesenen Karte und wird in den Controller injiziert.

```
angular.module('nfc.registration')
    .service("NfcRegistrationService", function () {
        var uid = "";
        return {
            getUid: function () { return this.uid; },
            setUid: function (id) { this.uid = id; },
            clear: function () { this.uid = ""; }
        };
    })
```

Listing 6.3: Der Datenspeicher der Registrierung besitzt manipulierende und sondierende Methoden für den Zugriff auf die definierte Variable.

6.1.2 Deregistrierung

In Abbildung 6.4 ist das Formular für die Lösung der Verknüpfung von Karten und Benutzerkonten dargestellt, das dem der Registrierung, bis auf das fehlende Feld der Seriennummer, ähnelt.

Nach dem Auslösen der Deregistrierung wird eine Anfrage an den Server abgeschickt. Die erhaltene Antwort wird dem Benutzer nach dem in Abbildung 6.3 dargestellten Farbmuster visualisiert.

Durch die Simplizität der Deregistrierung benötigt das Modul keinen Service und besitzt nur einen navigierbaren Zustand.

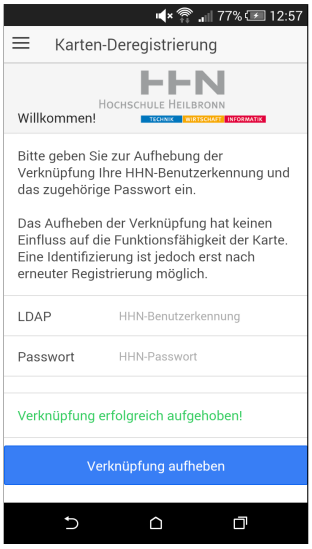

Abbildung 6.4: Formular der Deregistrierung

6.1.3 Allgemeiner Aufbau und andere Inhalte

Der allgemeine Aufbau der Applikation entspricht dem Modul-Entwurf aus Abschnitt 5.3.1. Einzig das Modul zur Nutzung der Ionic-Funktionen kam hinzu. Die Zusammenführung aller Module ist in Listing 6.4 dargestellt.

```
1 angular.module('hhn', ['ionic','nfc.registration','nfc.
      deregistration', 'hhn.constants'])
```

Listing 6.4: Eingebundene Module der Applikation

Als über die Ziele der Arbeit hinausgehende Inhalte wurden Seiten für Standardinhalte mobiler Applikationen realisiert. Diese umfassen eine Seite mit den verwendeten Frameworks und Plugins der mobilen Applikation, sowie deren Lizenzen. Darüber hinaus wurden Seiten für Impressum und Nutzer-Feedback erstellt, die vor dem Produktivbetrieb mit entsprechenden Inhalten ausgestattet werden müssen (siehe Abbildung 6.5).

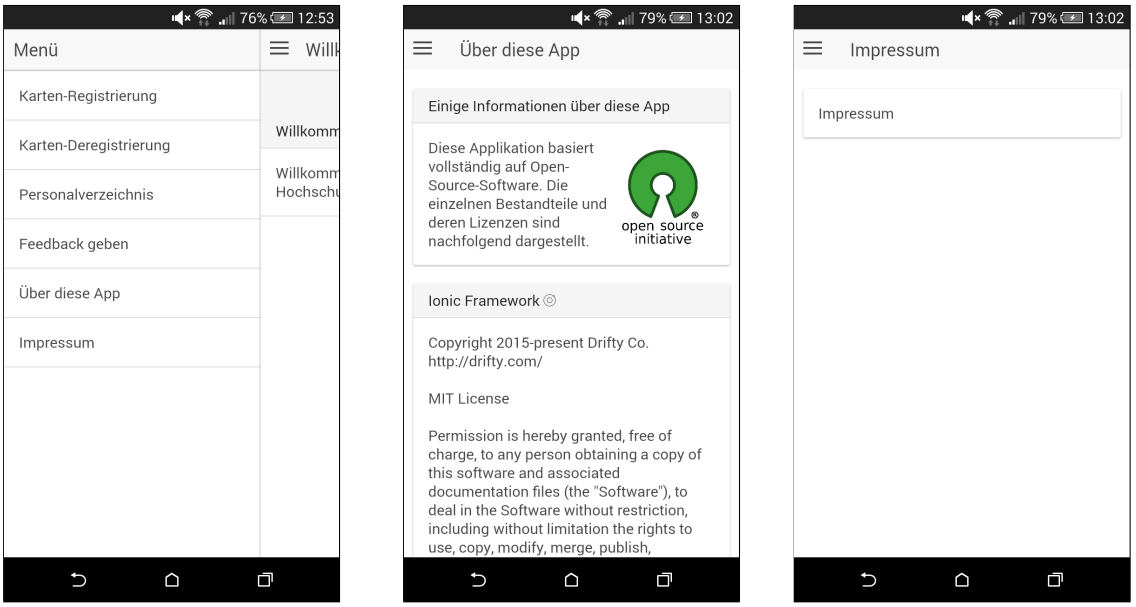

Abbildung 6.5: Menü der Applikation (links), Seite mit den verwendeten Frameworks und deren Lizenzen (Mitte) und einem Rumpf für das Impressum (rechts).

Um die Erweiterbarkeit der Modulstruktur zu testen, wurde ein von Tomi Semet[2] geschriebenes Ionic-Modul (siehe Anhang A) integriert. Die Integration wird in Abschnitt 7.1.4 dargelegt und diskutiert.

[2] Auszubildender des Rechenzentrums der HHN: `https://www.hs-heilbronn.de/tomi.semet`

6.2 Prototyp des Server-Backends

Das Server-Backend wurde nach dem in Abschnitt 5.2 abgebildeten Klassendiagramm (Abbildung 5.3) implementiert.

Listing 6.5 zeigt einen Ausschnitt des REST-Service der Anwendung. Das Behandeln der Requests der Clients beschränkt sich auf des Nutzen des `DelegateTagService` und Umwandeln von dessen Antwort in entsprechende HTTP-Antworten.

```
1  @Path("1.0/nfc")
2  public class TagServiceImplRest {
3    TagService service = null;
4      public TagServiceImplRest() {
5          service = new DelegateTagService();}
6
7     @POST
8     @Path("/update")
9     @Consumes("application/json")
10    @Produces("application/json")
11   public Response updateTag(UserTagImpl user) {
12     Result result = service.updateTag(user);
13     return handleResult(result);
14   }
15
16   private Response handleResult(Result r) {
17     switch (r.getStatus()) {
18     case 200: {
19           return Response.ok(r, "application/json").build();}
20     case 400: {
21           return Response.status(400).entity(r).build();}
22     default: {
23           return Response.serverError().entity(r).build();}
24         }
25   }
26 }
```

Listing 6.5: Behandeln eines POST-Request für die Registrierung von NFC-Tags durch Benutzung des `DelegateTagService`.

Listing 6.6 zeigt am Beispiel der Registrierung das Delegieren an einen injizierten Service mit anschließender Umwandlung der Antwort in ein für die App aufbereitetes Format. Das Verhalten besitzt große Ähnlichkeit zu Listing 6.5, ist jedoch komplett unabhängig von Nachrichtenprotokollen oder verwendeten Technologien und ermöglicht die Nutzung des Services mit jeder beliebigen Zugriffsschicht.

```
1 public class DelegateTagService implements TagService {
2   //constructor, service-injection
3     protected void handleUpdateTag(UserTag u) throws
          NfcLinkException {
4     service.addTag(u.getUsername(), u.getPassword(), u.getUid());}
5
6   @Override
7   public Result updateTag(UserTag user) {
8     Result result = null;
9     try {
10      handleUpdateTag(user);
11      result = new Result(200, "Verknüpfung war erfolgreich!");
12    } catch (NfcLinkException e) {
13        result = new Result(500, e.getMessage());
14    } catch (IllegalArgumentException v) {
15        result = new Result(400, "Eingabe ist ungültig!");}
16    return result;
17  }
18 }
```

Listing 6.6: Umwandlung der Service-Antwort in ein `Result`-Objekt nach Delegation an einen behandelnden Service. Vorgehensweise ist äquivalent für die Deregistrierung anzuwenden.

Für den Zugriff auf die Datenbank wird durch Abstraktion nur das Verhalten implementiert, ohne darunterliegende Technologien kennen zu müssen. Deren Spezifizierung erfolgt erneut durch Injektion mittels zentraler Instantiierung.

```
1 public class NfcServiceDatabase implements NfcService {
2   //constructor, service-injection
3   @Override
4   public void addTag(String username, String password, String uid)
5       throws NfcLinkException {
6     User user = service.getUser(username, password);
7     if (user != null) {
8       Tag tag = service.getTag(uid);
9       if (tag != null)
10        throw new NfcLinkException("Tag bereits verknüpft.");
11      tag = service.createTag(uid);
12      user.setTag(tag);
13      service.store(user);
14    } else {
15      throw new NfcLinkException("Benutzername oder Passwort ungü
            ltig");}
16  }
17 }
```

Listing 6.7: Vorgehen bei der Registrierung unter Verwendung einer Datenbank.

6.3 Sicherheitsmechanismen

Die CORS-Mechanismen verhindern einen Zugriff auf Ressourcen nicht gelisteter Quellen. Das Whitelisting dieser Domains erfolgt in der Cordova-Konfigurationsdatei der Applikation (siehe Listing 6.8). Zusätzlich werden in der `index.html` (siehe Listing 6.9) der Anwendung die Content Security Policy (CSP)-Einstellungen im Header hinterlegt. Da eventuell für die Applikation notwendige Skripte nicht lokal vorgehalten werden, müssen die Restriktionen für Skript-Quellen gesetzt werden, um XSRF-Angriffe vorbeugen zu können. Für den in Abschnitt 5.4 vorgestellten Kommunikationsfluss müssen alle Antworten des Servers über einen Filter (siehe Listing 6.10) mit CORS-Headern dekoriert werden.

```
1 <access origin="*"/>
2 <access origin="tel:*" launch-external="yes"/>
3 <access origin="mailto:*" launch-external="yes"/>
4 <allow-navigation href="*"/>
5 <allow-intent href="*" />
```

Listing 6.8: Whitelisting aller vertrauenswürdiger Ziele und Quellen in der `config.xml` der Applikation.

```
1 <meta http-equiv="Content-Security-Policy" content="default-src *;
      style-src 'self' 'unsafe-inline'; script-src 'self' 'unsafe-
      inline' 'unsafe-eval'">
```

Listing 6.9: Vertrauenswürdige Quellen für Medien und Skripte müssen im Header der `index.html` gesetzt werden.

```
1 @Provider
2 public class AccessControlResponseFilter implements
      ContainerResponseFilter {
3   @Override
4   public void filter(ContainerRequestContext requestContext,
        ContainerResponseContext responseContext) throws
        IOException {
5     MultivaluedMap<String, Object> headers = responseContext
6         .getHeaders();
7     headers.add("Access-Control-Allow-Origin", "*");
8     headers.add("Access-Control-Allow-Headers",
9         "Authorization, Origin, X-Requested-With, Content-Type");
10    headers.add("Access-Control-Expose-Headers",
11        "Location, Content-Disposition");
12    headers.add("Access-Control-Allow-Methods","POST,OPTIONS");}
13 }
```

Listing 6.10: Antworten des Servers müssen über einen Filter mit CORS-Headern dekoriert werden.

7 Fazit und Ausblick

In diesem Kapitel werden die Ergebnisse der Analyse, des Entwurfs und der Entwicklung diskutiert, sowie Optimierungs- und Erweiterungsmöglichkeiten der Applikation in Form eines Ausblicks vorgestellt.

7.1 Diskussion

Das Ziel der Arbeit war das Entwickeln einer mobilen Anwendung zur Registrierung von NFC-Tags für die Identifizierung von Personen. Nach vorausgehender Analyse der Anforderungen (Kapitel 3) und verfügbarer Technologien (Kapitel 4), sowie des Erstellens eines Entwurfs (Kapitel 5), wurde die Applikation realisiert (Kapitel 6). Dieses Ziel wurde nach Auffassung des Autors erreicht.

7.1.1 Analyse: Frameworks für hybride Applikationen

Für die Umsetzung der hybriden Applikation wurde Ionic Framework ausgewählt, welches im Zusammenspiel mit AngularJS und Phonegap nach den in Kapitel 3 aufgestellten Anforderungen bestmöglich entsprach. Die Analyse der Frameworks basierte dabei auf den Angaben der einzelnen Hersteller. Sicherlich könnte durch eine Test-Implementierung eine noch aussagekräftigere und zuverlässigere Auswahl getroffen werden. Es ist jedoch anzuzweifeln, dass der dabei gewonnene Mehrwert den entstehenden Aufwand in Einarbeitung und Umsetzung rechtfertigt.

Eine Vielzahl der Lösungen wurde auf Grund aufkommender Kosten bereits frühzeitig aus der Analyse ausgeschlossen. Es bleibt fragwürdig, ob die Bereitstellung finanzieller Mittel die Entwicklung der Applikation vereinfacht, oder sogar bessere Ergebnisse erzielt hätte. Der zusätzliche Funktionsumfang der kommerziellen Lösungen ergab sich meist aus Dienstleistungen wie z.B. Mobile Backend as a Service (MBaaS) oder 24h-Support, welche für die Umsetzung an der Hochschule niedrige Relevanz besitzen.

Die Anzahl auf dem Markt verfügbarer Lösungen für hybride Apps steigt stetig. Für die Analyse wurden daher nur bereits stabil verfügbare und verbreitete Frameworks betrachtet. Es bleibt abzuwarten, welche Lösungen sich in naher Zukunft auf dem Markt etablieren werden. Es ist jedoch anzunehmen, dass Ionic Framework auf Grund seines Funktionsumfangs und der Community eines dieser Frameworks sein wird.

7.1.2 Analyse: JAX-RS Implementierungen & JSON-Umwandlung

Als Implementierung des JAX-RS 2.0 Standards wurde RESTEasy ausgewählt. Die Einfachheit in Einrichtung und Benutzung, sowie die vielfältige Verfügbarkeit von Providern für Datenformate harmoniert mit der niedrigen Komplexität des zu entwickelnden Service. Die Auswahl basierte auf einer vergleichbaren Beispiel-Implementierung. Die über den Standard hinaus funktionsreicheren Frameworks könnten genauso gut

verwendet werden. Es ist aber offensichtlich, dass eine möglichst einfache Lösung gewählt werden sollte, um nicht künstlich zusätzliche Komplexität zu schaffen.

7.1.3 Entwurf Server-Backend

Der Entwurf des Server-Backends ist so konzipiert, dass der Wechsel der verwendeten Technologie von Zugriffsschicht und Datenhaltung ohne umfassende Änderungen an der Code-Basis durchführbar sind. Durch die dafür eingeführten Abstraktionen ergibt sich zwar die angesprochene Flexibilität in der Wahl der Frameworks, sowie eine erhöhte Wiederverwendbarkeit der tieferen Schichten, dennoch sinkt die Verständlichkeit des Entwurfs bei Erweiterung. Zudem sei in Frage gestellt, ob für weitere Funktionalität der mobilen Applikation das bestehende Backend erweitert, oder besser modulspezifische Server-Anwendungen entworfen werden sollten, die isoliert entwickelt, betrieben und gewartet werden können.

7.1.4 Funktionaler Prototyp

Der funktionale Prototyp der Anwendung enthält alle geforderten Funktionen zur Registrierung und Deregistrierung von NFC-Karten und integriert darüber hinaus weitere in Applikationen enthaltene Inhalte, wie z.B. eine Auflistung der verwendeten Frameworks und Plugins mit Lizenzen. Die Applikation ist auf der Android-Plattform lauffähig und kann innerhalb des Hochschulnetztes mit einer Testdatenbank getestet werden. Für den Produktivbetrieb muss jedoch die Übertragung der Daten über eine sichere HTTPS-Verbindung konfiguriert werden, um eine Übermittlung im Klartext zu verhindern. Im Rahmen der Entwicklung wurde auf die Konfiguration verzichtet. Die Oberfläche der Applikation mit ihren Inhalten ist als Prototyp zu betrachten, ist nicht internationalisiert und erfüllt keine Usability-Normen[1].

7.1.5 Modularisierung des Prototyps

Im Folgenden wird der Erfolg der Modularisierung der Applikation diskutiert. Hierfür wurde ein von Tomi Semet[2], Auszubildender des Rechenzentrums der HHN, geschriebenes Ionic-Modul (siehe Anhang A) integriert .

Die Ressourcen des Moduls waren nach Typ der Dateien, und nicht nach Funktionalität sortiert. Diese mussten daher erst in die funktionsorientierte Modulstruktur gebracht werden (vgl. Abb. 7.1). Dafür wurde ein Verzeichnis für alle Ressourcen des Moduls angelegt, in welches die HTML- und JavaScript-Dateien, sowie verwendete Bilder integriert wurden. Referenzen auf Bilder mussten korrigiert werden. Das Modul war zuvor nicht in ein Menü eingebettet gewesen, sodass in den HTML-Seiten der Code

[1] DIN EN ISO 9241-110 beschreibt z.B. die Grundsätze der Dialoggestaltung
[2] `https://www.hs-heilbronn.de/tomi.semet`

minimal angepasst werden musste. Die interne Struktur des Moduls mit Controllern, Navigation und Moduldefinition konnte identisch übernommen werden.

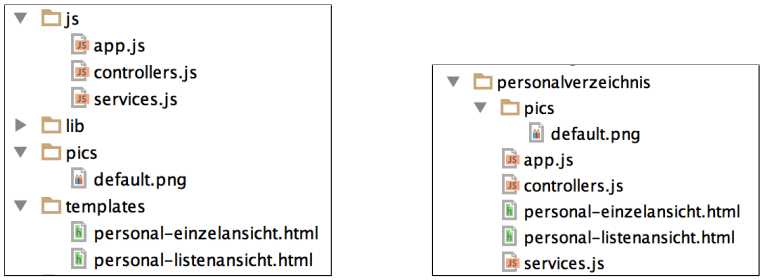

Abbildung 7.1: Die Strukturänderung bei der Integration des Moduls.

Nach der Aufbereitung erfolgte die eigentliche Integration des Moduls. Dabei wurden alle JavaScript-Ressourcen in die `index.html` aufgenommen. Zusätzlich musste die Referenz auf das Personenverzeichnis in der Moduldefinition der Applikation ergänzt werden, sowie ein passender Menüeintrag erstellt werden. Für die Verwendung der Telefon- und E-Mail-Funktion des Smartphones mussten die Verweise unter Verwendung externen Applikationen in der `config.xml` freigeschaltet werden.

Die Integration des Moduls wirkt durch die zusätzliche Umstrukturierung komplex, war jedoch absehbar einfach. Für die Integration des Moduls wurden insgesamt 8 Zeilen Code hinzugefügt, sowie die Moduldefinition um den Modulnamen erweitert.

7.2 Ausblick

Vor einer realen Nutzung des Systems müssen der Datenschutz und andere organisatorische und rechtliche Pflichten, wie zum Beispiel das Telemediengesetz berücksichtigt werden. Die Umsetzung der dort geregelten Pflichten durch die Entwickler wird von den Betreibern der App-Stores erwartet. Auch der Aufwand des Ausrollens in einen Store darf nicht unterschätzt werden. Dieser kann durch die Umsetzung organisatorischer Maßnahmen zum automatischen Testen und kontinuierlichen Ausrollen der hybriden Applikation reduziert werden und ermöglicht gleichzeitig die effektive Entwicklung im Team.

Eine Benutzerevaluation oder Gebrauchstauglichkeitsnormen entsprechende Umstrukturierung der Anwendung ist ebenso denkbar, wie ein Testbetrieb mit verschiedenen Identifizierungssystemen. Möglicherweise wird in naher Zukunft der NFC-Sensor für die iOS-Plattform nutzbar, sodass die Applikation auch dort Anwendung finden kann.

Mit zusätzlichen Modulen, wie z.B. einem personalisierten Stundenplan oder der Integration des Speiseplans der Mensa, gibt es nach Meinung des Autors einige Anreize, die Applikation zu erweitern und den Produktivbetrieb anzustreben.

8 Literaturverzeichnis

[1] ISO, "Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)," ISO/IEC 18092:2013, International Organization for Standardization, Geneva, Switzerland, 2013.

[2] NFC Forum, "NFC and Contactless Technologies." `http://nfc-forum.org/what-is-nfc/about-the-technology/`. – Letzter Zugriff: 16.06.2015.

[3] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures." `https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`, 2000. – Letzter Zugriff: 16.06.2015.

[4] The World Wide Web Consortium (W3C), "Cross-Origin Resource Sharing - W3C Recommendation: 16 January 2014." `http://www.w3.org/TR/cors/`. – Letzter Zugriff: 24.07.2015.

[5] D. Schadow, *Java-Web-Security: Sichere Webanwendungen mit Java entwickeln*. Heidelberg: Dpunkt.Verlag, 1. ed., 2014.

[6] A. Freeman, *Pro AngularJS*. Apress Media LLC, 2014.

[7] Apache Cordova, "About Cordova." `https://cordova.apache.org/#about`. – Letzter Zugriff: 16.06.2015.

[8] PhoneGap, "About the Project." `http://phonegap.com/about/`. – Letzter Zugriff: 16.06.2015.

[9] T. Bosch, S. Scheidt, and T. Winterberg, *Mobile Web-Apps mit JavaScript : Leitfaden für die professionelle Entwicklung*. Entwickler Press, 2012.

[10] Appcelerator Inc., "The Appcelerator Platform." `http://www.appcelerator.com`. – Letzter Zugriff: 06.07.2015.

[11] Embarcadero Technologies Inc., "Das FireMonkey-Framework." `https://www.embarcadero.com/de/products/rad-studio/firemonkey`. – Letzter Zugriff: 06.07.2015.

[12] Intel Corporation, "Intel XDK." `https://software.intel.com/en-us/intel-xdk`. – Letzter Zugriff: 06.07.2015.

[13] NativeScript, "Build truly native apps with JavaScript." `https://www.nativescript.org`. – Letzter Zugriff: 06.07.2015.

[14] Sencha Inc., "Create native-looking HTML5 apps using JavaScript." `http://www.sencha.com/products/touch/`. – Letzter Zugriff: 06.07.2015.

[15] WebMynd Inc. d/b/a Trigger, "Trigger.io - The simplest way to build amazing mobile apps." https://trigger.io. – Letzter Zugriff: 06.07.2015.

[16] Xamarin Inc., "Mobile App Development & App Creation Software - Xamarin." http://xamarin.com. – Letzter Zugriff: 06.07.2015.

[17] AppGyver Inc., "Supersonic Framework." http://www.appgyver.com. – Letzter Zugriff: 06.07.2015.

[18] Famous Industries Inc., "Famous Engine." http://famous.org. – Letzter Zugriff: 06.07.2015.

[19] Drifty Corporation, "Ionic: Advanced HTML5 hybrid Mobile App Framework." http://ionicframework.com. – Letzter Zugriff: 06.07.2015.

[20] The jQuery Foundation, "A Touch-Optimized Web Framework." https://jquerymobile.com. – Letzter Zugriff: 06.07.2015.

[21] Telerik, "Kendo UI - Everything for building web and mobile apps with HTML5 and JavaScript." http://www.telerik.com/kendo-ui. – Letzter Zugriff: 06.07.2015.

[22] Asial Corporation, "Onsen UI - The Answer to Cordova UI Development." http://onsen.io. – Letzter Zugriff: 06.07.2015.

[23] Apache Software Foundation, "Apache CXF: An Open-Source Services Framework." http://cxf.apache.org. – Letzter Zugriff: 07.07.2015.

[24] Oracle Corporation, "Jersey - RESTful Web Services in Java." https://jersey.java.net. – Letzter Zugriff: 07.07.2015.

[25] Red Hat Inc., "RESTEasy." http://resteasy.jboss.org. – Letzter Zugriff: 06.07.2015.

[26] Restlet Inc., "Restlet Framework: Build Web APIs in Java." http://restlet.com/products/restlet-framework/. – Letzter Zugriff: 06.07.2015.

[27] Google Inc., "Versioning Your Applications." http://developer.android.com/tools/publishing/versioning.html. – Letzter Zugriff: 12.07.2015.

[28] T. Igoe and D. Coleman, *NFC mit Android und Arduino: near field communication für Maker*. O'Reilly, 2014.

Anhang

A Ionic-Modul: Personalverzeichnis

Das Ionic Modul ist für das Personalverzeichnis der Mitarbeiter und Professoren der Hochschule Heilbronn konzipiert. Es kommuniziert über einen REST-Service mit dem LDAP-Server der Hochschule Heilbronn.

Das Modul besitzt zwei Ansichten, wobei sich in der ersten Ansicht die Liste aller Mitarbeiter der Hochschule mit Suchfunktion zur Filterung nach Vor- und Nachnamen findet. Durch Auswahl eines Mitarbeiters gelangt man zur Einzelansicht. Dort ist das bei der Hochschule hinterlegte Bild und die Kontaktdaten der Person hinterlegt.

Durch die Auswahl der Telefonnummer oder E-Mail-Adresse erfolgt die Kontaktaufnahme.

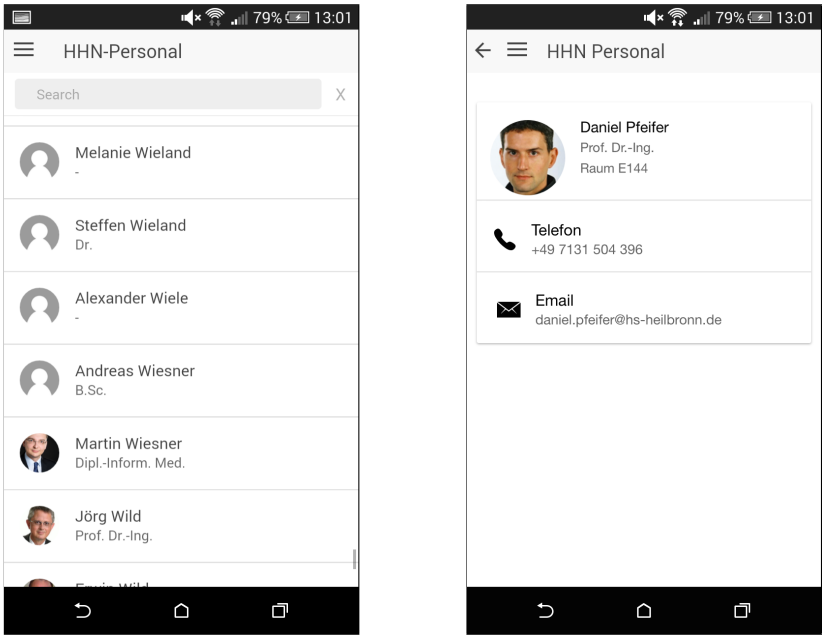

Abbildung A.1: Listen- und Personenansicht des Personalverzeichnisses.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

(Ort, Datum) (Unterschrift)